Studies on the Design of Minimal Logic Networks(Dissertation_全文)

AUTHOR(S):

Ibaraki, Toshihide

CITATION:

Ibaraki, Toshihide. Studies on the Design of Minimal Logic Networks. 京都大学, 1970, 工学博士

ISSUE DATE:

1970-09-24

URL:

https://doi.org/10.14989/doctor.r1669

STUDIES
ON
THE DESIGN OF MINIMAL LOGIC NETWORKS

NOVEMBER 1969

TOSHIHIDE IBARAKI

STUDIES

ON

THE DESIGN OF MINIMAL LOGIC NETWORKS

NOVEMBER 1969

TOSHIHIDE IBARAKI

STUDIES

ON

THE DESIGN OF MINIMAL LOGIC NETWORKS

by

TOSHIHIDE IBARAKI

Submitted in partial fulfillment of the
requirement for the degree of

DOCTOR OF ENGINEERING

(Applied Mathematics and Physics)

at

KYOTO UNIVERSITY

KYOTO, JAPAN

November 1969

Preface

The history of computers may way back to Leibniz's "calculator" or Babbage's "analytic engine". Actual applications of computers and their enourmous expansion, however, had to wait until the introduction of modern electronic computers, initiated by MARK I and ENIAC in the middle of 1940's. These pionnering machines were followed by rapidly increasing number of more powerful machines, which might be well classified by the first generation, the second generation and the third generation computers.

This thesis is devoted to explore a certain aspect of the computer design.

As might be easily noticed, a clear distinction berween the machines in the early days, which might be credited to the personal enthusiasm or to the inspiration of a genius, and the modern computers is that the size of the latter has grown so large that the involvement of a large number of people is required to complete the design of a machine. Since the emergence of the computer, therefore, a considerable amount of effort has been directed to mathematically formulate the design process and possibly to automate its entire steps. A phase of the design process called the logic design, which determines the detailed logic structures among logic components in a machine, has received a strong attention, because it contains formidably tedious procedures and yet appear to allow its mechanization.

An important initial step towards this goal is established by Hanzawa-Nakajima and Shannon by introducing Boolean algebra as a mathematical tool for accomplishing the logic design. The Quine-McCluskey method for designing a minimal AND/OR two level networks would be a main result of this approach. Under some conditions, this method yields optimal (minimal) networks in a certain sense. The minimality of a network is an essential factor for obtaining a fast and economical machine, and also of theoretical importance.

It was a reasonable consequence to introduce the use of computers to perform or aid the design process of computers, as the rapid increase of the size is incurred. Even with the aid of computers, since the Quine-McCluskey method or any other methods for optimal networks are not capable of dealing with the entire machine directly because of its prohibitingly large amount of computation involved in performing the design, the present approaches commonly adopted are heuristic ones assisted by these Boolean algebraic methods. Therefore, at present, the optimality of the entire network is not usually attained, though the local optimality may be sometimes guaranteed by use of the aforementioned Boolean algebraic method.

Furthermore, with the advent of the explosive development of recent technology, in particular IC and LSI technology, the assumptions and restrictions in the logic design continue to evolve, allowing new capabilities, introducing new circuit restrictions and greatly

increasing the logic complexity containable in a unit volume. As a result, those methods which have been standard for some time became obsolete and useless. Even the classical methods represented by the Quine-McCluskey's do not seem to catch up with these innovations. The new situation needs new approaches, i.e., optimal network design methods for such new types of networks, which may be applicable at least to local small networks.

In view of the current status as mentioned above, this thesis is devoted to create new methods which may partially accept this new situation.

The topics discussed in the thesis are roughly devided into three categories: (1) design of networks of threshold gates, (2) design of networks of negative gates and (3) design of various networks by means of integer programming. The first and the second problems have become important since the recent electronics technology enabled to manufacture considerably complex logical gates such as threshold gates and negative gates with enough reliability. The design problems with these gates are difficult since the functional form of each gate is also subject to the designer's specification within the gate type of threshold functions or negative functions, as well as the structure of the network.

For the first problem the functional properties of a threshold function (completely monotonic function) are investigated and they are utilized to yield an optimal or nearly optimal network. For the second

problem, an algorithm for optimal network is developed such that a network with the minimum number of gates is obtained.

The integer programming formulation can be applied to various types of networks with conventional gates such as AND, OR, NAND, NOR, EXCLUSIVE OR, and their combinations. Theoretically, the formulation can be further extended to even sequential networks including the state assignment problems. Merits of this approach consist in its flexibility that a wide variety of circuit restrictions such as fan-ins and fan-outs restrictions, can be easily incorporated and in its versatility that different networks and objectives can be optimized by simply replacing the constraint sets and the objective function in the integer program. The computational feasibility in this direction was tested for NOR and NOR-AND networks. The results appear promising and encourages further investigation.

The author would like to hope that the properties and algorithms discussed in the thesis will help to move the status of the logic design one step forward.

Contents

Chapter 1. Introduction

1.1 Problems in logic design

The design procedure of a modern electronic computer, or any other digital machine, may be devided into two main phases: (1) preconstruction analysis and (2) logic design and its implementation. The first is, in other words, the system design of a machine under consideration as well as its analysis and verification. The second is concerned with the specification of detailed structure of each subsystem which is defined in the first phase. The term "detailed structure" includes the logic structure, i.e., the connections among logic gates, delay element, flip-flops and other components, to realize the given function of the subsystem, and the physical structure, i.e., the assignment of logic to cards, the assignment of cards to backboards, and the routing of these backboards and cards.

For an ordinary digital computer, the first phase consists of decomposing the entire system into several smaller subsystems such as memory unit, arithmetic unit, control unit and I/O (Input-Output) unit, and then specifying global structure of each subsystem, such as what kind of memory devices may be used and their hierarchy, how many registers should be available in the arithmetic unit, what kind of logic gates and flip-flops may be employed in the control unit, and so forth.

Main goal of the entire design process is to attain a certain optimality defined by the speed, yielding the fastest machine, by the cost, yielding the most economical machine,

by the ease of use, by its versatility, or by whatever the designer wants to optimize, under some restrictions such as the maximum cost, available memory devices, the speed of I/O equipments, etc..

It is not easy to design a machine which works under given specification and, yet, achieves such optimalities because of its formidable complexity. This is the motivation of introducing the computers into the design process. Considerable amount of effort has been made to automate each design process, or at least to aid the process (computer aided design).

The present status of design automation or, more moderately, computer aided design may be seen in Breuer[113]. According to this survey paper, in the first phase of the design, computers are utilized to simulate a tentatively designed system, using simulation languages such as SIMSCRIPT, GPSS and SOL. Based on the analysis performed by computers, the final decision may be made by men. The second phase is automated in much higher degree. However, most of algorithms used in each step of the design are heuristic since no practical exact algorithm which can deal with such large scale combinatorial problems as encountered in the design process is available. This means that the optimality is abondaned from the outset.

This thesis is concerned with the logic design in the second phase of the whole design process and attempts to provide new mathematical tools for designing optimal or near optimal networks. Although all the theories presented in the thesis may not be powerful enough to realize globally

optimal networks, it is believed that at least they can be applied locally and provide basis for further development of optimality design.

In designing the logic structure of each subsystem (network), i.e., in determining logic gates, flipflops, etc., and their connections,the first difficulty is the large sizes of such networks. The process of logic design necessarily involves the computation of combinatorial nature, which may prohibit the application of such theories because of the excess amount of computation required.

One approach to overcome this difficulty is again to decompose the whole network into small subparts so that each subpart may allow its optimal design. In other words, the total function is split into many simple subfunctions, such that each subpart for a subfunction can be realized with optimality and, the aggregate of those subparts properly interconnected works as the required network. Assuming that each subpart satisfies the optimality condition, their aggregate would be close to the total optimality if an appropriate decomposition of the network is performed. Of course, the "appropriate" decomposition should be defined mathematically and should be studied more carefully. At present, although a number of attempts and proposals are known for that purpose, there seems to be no rigorous discussion on the subject. In other words, most of proposals are of heuristic nature and therefore can not prove the optimality of the resulting decompositions, though considerably good decompositions may usually be obtained

in a certain sense. In any event, this approach will provide us with a suboptimization technique of a large network.

With this method, it is not an oversimplification to state that the logic design could be reduced to designing logic functions, using prespecified logic gates, which are subject to some network constraints imposed by engineers.

In this thesis, therefore, the discussion will be confined to the design of given logic (switching) functions. Even with this simplified argument, however, the present status of logic design is still not satisfactory. It is because the recent development of electronics such as integrated circuit and large scale integration keeps on increasing new capability of each gate and, at the same time, complicating the situation by introducing new network restrictions. As a consequence, in many cases, the exhaustion of all possible networks is only available method to obtain optimal networks for given logic functions.

To meet these advances in switching circuit technology, this thesis considers the following several factors which appear prominent and essential among others. Algorithms to solve part of these new problems are developed and proposed. These algotithms deviate from the classical switching theory which has been a main mathematical tool of logic design.

(1) New types of gates became practically available and commonly manufactured. For example, NOR and

NAND gates are the most popular gates, whereas the classical minimization theory by Quine-McCluskey is only for AND/OR gates. Another example is those gates which are of certain types but the concrete functional form of each gate is subject to the designer's determination. Consider a threshold gate. This gate can represent any function known as a threshold function by adjusting the values of weights and a threshold associated with the gate. Thus the determination of weights and a threshold is also a task of logical designer. Also, MOS integrated circuits can represent rather complex negative functions. It should be a good approximation to assume that a gate can represent any negative function. Thus the functional form of each gate is also subject to designer's specification.

(2) Network restrictions should be taken into consideration. In order to physically realize a reliable logical circuit, some restrictions are usually imposed, such as the maximum number of inputs (fan-ins) to the gate, and the maximum number of outputs (fan-outs). In some cases, more complicated restrictions such as the maximum length of parallel wires are also imposed, depending on the actual circuits under consideration.

(3) The optimality criteria may vary according to the components employed in realizing the network.

> For instance, if the transistors and diodes are used, the total number of these components constitutes a major part of the total cost, and if integrated circuits or large scale integration circuits are used, the number of connections would be more important. In some cases, the number of layers, which is relevant to the number of crossovers among connections, almost determines the total cost. The number of levels of gates in the designed network is an essential factor in limiting the speed of logic operation. The speed might be more important than the cost. Minimization algorithms investigated so far has been dealing with only the minimization of the number of gates in the network, with almost no exception. This is partly because of the lack of correct recognition of the logic design, and partly because of its difficulty of handling other objectives.

In the following in Chapter 1, Quine-McCluskey theory of minimization will be first sketched to show its limitation. It is also pointed out that certain situation can be handled by just amending the classical theory. However, all the topics discussed in this thesis seem to refuse the discussion in the framework of the classical theory. Therefore, new approaches have to be taken to accept these new situations. Each approach will be briefly mentioned in this chapter.

1.2. Quine-McCluskey Method and its Limitation

Quine-McCluskey method was proposed in 1952 by Quine[87] and in 1956 by McCluskey[66]. It has been the only systematic method synthesizing optimal networks since then. It assumes a 2-level network, AND gates in the first level and an OR gate in the second level, as illustrated in Fig. 1.1. The OR gate in the second

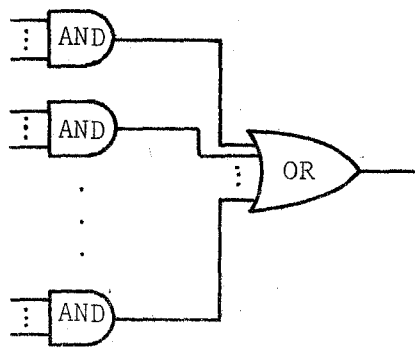


Fig.1.1 2-level AND/OR network.

level can receive inputs from external variables and outputs of the gates in the first level, while AND gates in the first level can receive inputs only from external variables. External variables assumed here are variables $x_1, x_2, \ldots, x_n$ and their negations $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$. Each gate has no restrictions on the number of inputs.

Under these assumptions, the minimization of the total number of gates is equivalent to finding a minimal cover of prime implicants of $f(X)$, a logic function to be realized, according to the Quine-McCluskey theory. In other words, each prime

implicant of f(X) corresponds to an AND gate in the first level, thus finding a minimal cover implies to define a minimal set of AND gates which can collectively be ORed to realize f(X). Since considerably large covering problems to find minimal covers can be practically solved with various methods [66][5][48], the Quine-McCluskey approach is an effective tool for the problem.

Now consider, however, the case in which the maximum number of inputs (fan-ins) of each gate is specified. Then, generally speaking, the number of levels of network for f(X) will exceed two, thus prohibiting the direct application of the Quine-McCluskey method. Also, the case in which gates in the network are different from AND and OR will neccessiate the amendments of the Quine-McCluskey method, even without the fan-ins restrictions.

A successful modification in the latter direction was established by Gimpel[31], when the design of NOR (or NAND) gate networks is concerned. It is known that a three level NOR network can realize any logical function f(X), if the fan-ins restrictions are not imposed. A typical network is illustrated in Fig. 1.2.

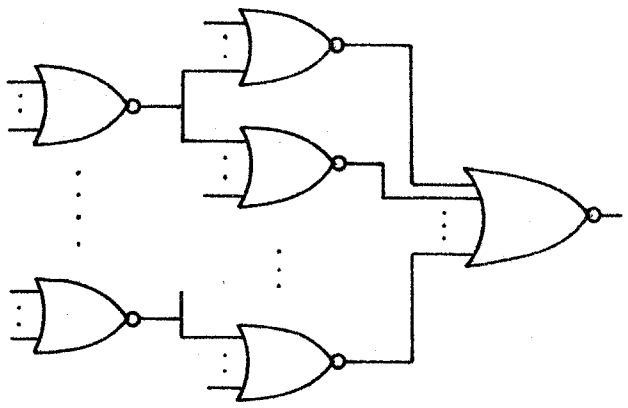


Fig. 1.2. 3-level NOR network.

Each gate can receive inputs from external variables and the outputs of gates in the preceding level. External variables considered here are $x_1, x_2, \ldots, x_n$ only, excluding their negations.

Extending the Quine-McCluskey theory, Gimpel reduced the problem of minimizing the total number of NOR Gates to finding a minimal cover of CC-table of prime permissible implicants, which were introduced by him[31].

Although numerous attempts have been made for other cases, especially when the fan-ins restrictions are imposed, there seems to exist no minimization method which is not only theoretically complete but also practically applicable, to the author's knowledge.

Moreover, the situation is further compounded when other optimal criteria than the minimization of the number of gates are adopted. Also combination of different objectives sometimes reflects the real situation more appropriately, such as first minimizing the number of gates and then, among those networks with minimum number of gates, finding networks with the minimum number of interconnections.

Talking about the criterion in term of the number of levels, the minimum number is often known from the theoretical point of view, if no fan-ins restrictions are imposed. AND/OR networks and NOR networks mentioned above are such examples. With fan-ins restrictions, there seems no systematic method known to date.

In case of combination of different optimality criteria, the following procedure is commonly taken, if the exhaustion of all the optimal networks under the primal optimality

condition is practically possible. That is to search the optimal networks under the secondary criterion, among those which are optimal under the primal criterion. According to this approach, it is not hard to obtain all the optimal AND/OR and NOR (NAND) networks under a two-fold optimality criterion, if the primal objective is the number of gates.

However, other configurations of optimality criteria or those cases combined with fan-ins restrictions, etc, seem difficult to handle by the classical methods.

Contrary to the Quine-McCluskey approach, the integer programming formulation described in this thesis permits a wide variety of optimality criteria and network restrictions, if the function of each gate can be represented by integer linear inequalities. Conventional gates such as AND, OR, NOR, NAND, and EXCLUSIVE OR fall in this class. Optimality criteria, such as the number of gates, the number of connections and the number of levels, or their combination, can be handled within the framework of integer programming formulation. Also, the network restrictions such as fan-ins restriction and fan-outs restriction can be easily included in the formulation.

The branch and bound method[19] shares some similarlity with the integer programming formulation in special cases, and these two methods are compared in Chapter 6.

Those gates, whose actual functional forms are subject to the designer, are also difficult to deal with by means of the classical method. These problems and historical background are sketched in the next two sections.

1.3. Networks of Threshold Gates

The importance of threshold logic was recognized among people about 1958-60, through the study of core logic[84][98] and parametrons.[70][37] A typical threshod gate has n inputs with which their weights are associated, and a threshold. By varying the values of weights and threshold, a gate can represent various functions. For the design of networks of threshold gates, Quine-McCluskey type approach appears almost useless, because the determination of the actual functional form of each gate is involved in the design process.

In the early stage, most effort was directed to clarifying the class of functions which can be realized by threshold gates (threshold functions) and how to determine its structure for such function.[13][71][72][98][27] As a consequence of these investigations, it may be said that testing and synthesis procedures of a threshold function was well established. However, the status of the design of threshold gate network for non-threshold functions is far from satisfactory.

According to the author's observation, network synthesis methods presently available will fall into three categories. The first is to regard a threshold gate as a conventionally well-known gate, such as AND, OR, NOR and NAND, since they are all special cases of threshold functions. Then design a network with those simplified gates. Some improvement steps would be incorporated to obtain a better network.[70][27][20] Inherent from its heuristic nature, usually the resulting

networks would be far from optimal, under most of cost criteria,though it might be easy to carry out. This is because it ignores the fact that a single threshold gate can realize a considerably complicated function compared with these conventional gates.

The second is based on integer programming formulation. This area was the first in logical design, to which the application of integer programming was considered.[10][77] The design of threshold gate networks is coverted to optimizing a linear objective function subject to linear constraints, in which some variables are constrained to be integers. Accordingly, if we could solve this integer programming problem, an optimal network was obtained. The size of the problem, however, is usually prohibitingly large for the algorithms available to date. Although there is some possibility that this approach becomes a main stream in this field, it entirely depends on the future development of integer programming algorithms.

The last approach is based on the asummability theorem that a function is threshold function if and only if it is asummable.[13][27][22] Thus this method is to augment the functions with extra variables in order to modify a summable function equivalently into an asummable function, if we also consider the extra variables as input variables, as well as original external variables. A procedure which will result in networks with the minimal number of gates is known[94]. However this algorithm also involves a large amount of computation to yield an optimal solution, because of

its exhaustive nature. At present, this direction seems impractical.

A compromise was proposed[45] to facilitate the computation, by employing completely monotonic functions instead of threshold functions directly. A function is completely monotonic if and only if it is 2-asummable, which is much easier than asummability to handle. A threshold function is a completely monotonic function but not conversely. However, for functions of up to 8 variables, the complete monotonicity is a necessary and sufficient condition for the threshold realizability, and also remains as a rather tight necessary condition for larger functions. Thus a network of completely monotonic gates is also a network of threshold gates, with few exceptions.

Along this line, new properties of completely monotonic function are investigated in this thesis. Based on the results obtained, a new design procedure is devised. It is easy to perform, and still yield considerably economical networks, if not optimal.

There are also other numerous approaches in this area.[62][20][76] From the author's point of view, they are either impractical to perform or result in poor networks.

In most design procedures mentioned above, only the number of gates in the network receives attentions. Other objectives such as the total weight or the value of threshold appears difficult to handle, though they have a

significant effect on the reliable operation of gates[76][47]. Contrary to this, the integer programming approach can handle these objectives in principle, though computational experience is not available. This approach is only exception in this respect, to the authors knowledge.

1.4 Networks of Negative Gates

The study of networks of negative functions is motivated by the recent development in integrated circuit technology. An interesting fact is that a MOS gate is able to represent a considerably complex negative function because of its inherently high impedance. For example, a MOS gate which represents any negative function with up to 32 literals in the functional form as:

$$g = \overline{x_{11}x_{12}\cdots x_{1k_1} \vee x_{21}x_{22}\cdots x_{2k_2} \vee \cdots \vee x_{s1}x_{s2}\cdots x_{sk_s}}$$

can be manufactured[97]. As a consequence it is not too far from the reality to assume that any negarive function can be realized with a single gate, as far as a moderate number of input variables are considered.

The design of networks of negative gates for non-negative functions suffers from the same difficulty as the networks of threshold gates in the sense that actual functional form of each gate is subject to the designer's determination.

To the author's knowledge, there has not been any systematic approach to this problem. In this thesis, an algorithm to obtain a network with the minimum number of negative gates for a given function is presented, under the assumption that;

(1) The network consists of two levels.

(2) No fan-ins restriction on each gate is imposed.

The procedure can be extended to multiple output network design.

The problems under other assumptions such as networks with fan-ins restriction or networks with more than two levels are beyond the scope of this thesis. Also the problems with other types of objectives such as the number of interconnections or the number of transistors in use seem difficult to attack, though some attempts are made.[50]

1.5 Outline of the Thesis

As explained in earlier sections, topics in the thesis are confined to the problems in logical design which are difficult to handle with the classical Quine-McCluskey approach.

Chapters 2 and 3 deal with the design problems with threshold gates putting stress on the completely monotonic functions rather than threshold functions themselves. Chapter 2 examines properties of completely monotonic functions. Especially, a new concept "mutual monotonicity" is introduced. Let us denote the function f with the variable x_j specified to 1 or 0 as f_j or $f_{\bar{j}}$ respectively. f can be expanded as

$$f = f_j x_j \vee f_{\bar{j}} \bar{x}_j .$$

Then, f is completely monotonic if and only if f_j and $f_{\bar{j}}$ are both completely monotonic and, moreover, mutually monotonic. Expanding the original f as well as resulting expanded functions successively, we will obtain a "expansion diagram" of f. f is, therefore, completely monotonic if and only if all the pairs of functions in the expansion diagram are mutually monotonic.

Since the mutual monotonicity can be easily examined by comparing Boolean expression of functions, the above method for checking complete monotonicity seems promising. As an application of these concepts, the functional form of a completely monotonic function was investigated and solved.

When f is not completely monotonic, its expansion diagram possesses non-mutually monotonic pairs of functions. In Chapter 3, an algorithm is presented to remove these non-mutually monotonic pairs by adding an extra augmented variable to each such pair in the expansion diagram. The algorithm includes a procedure to provide these augmented variables by a combination of completely monotonic functions, thus resulting in a network of completely monotonic functions realizing a given function f. The theory can be extended to the case in which threshold functions are treated instead of completely monotonic functions.

In Chapter 4, an algorithm which yields networks with the minimum number of negative functions for a given function is developed. First,basic properties of a negative function are examined including both completely specified and incompletely specified cases. Using these results, the design procedure is converted to finding a minimal cover of maximal compatible sets, which are also defined in the same chapter. The set covering problems have been studied by various people since it can be applied in a number of fields. At present, considerably large problems can be solved in reasonable amount of computation time[67][29][48]. The generation of all the maximal compatible sets is also rather straightforward. Thus with this algorithm, reasonably large problems are expected to be solved. It is suitable for computer program, though no program was yet run on the computer.

Chapter 5 and 6 are concerned with the integer programming formulation of the network design problem, possibly under the network restrictions such as fan-ins restrictions and fan-outs restrictions, and also under a variety of cost criteria. In Chapter 5, NOR networks (NAND networks)and the networks in which mixture of different types of gates such as NOR and AND is used, are discussed. The formulation can also be extended to multiple outputs cases. Also, the design of sequential networks, including state assignment, can be handled along this line.

Thus by solving the resulting integer programs, we will have optimal networks in various definitions.

Chapter 6 briefly describes the effort directed to the development of an efficient code of integer program. In particular, the NOR network design is investigated in detail, because it may reveal an aspect of how to improve the integer programming code by utilizing inherent structures of the problem. The final code shares a certain similarity with the branch and bound approach by Davidson[19]. All the optimal NOR networks for functions of 3 variables are exhausted, under the fan-ins and fan-outs restrictions. Also all the minimal networks of NOR and AND gates (mixture) are computed. The latter networks are listed as a catalogue in Appendix. The optimality criterion used for the catalogue is to find networks with the minimum number of connections among those with the minimum number of gates.

Only existing algorithm for this types of "difficult" problems has been to exhaust all the possible networks and

then find optimal networks among those[44]. Since the integer programming approach (or possibly the branch and bound approach) seems much faster to solve, and yet flexible enough, it will play an impotrant roll in the future design automation application.

Before proceeding to details, it should be mentioned that the material descussed in Chapter 2 is mainly taken from published papers [106][108][105], Chapter 3 also from published papers [107][109], Chapter 4 from a report of the Department of Computer Science, University of Illinois, [50], Chapter 5 from reports of the same department, [80][81], and Chapter 6 also from reports of the same department, [48][7].

Apart from the acknowledgment which will be given later, it is noted here that papers [105][106][107][108][109] are the results of the work with Associate Prof. S.Yajima of Kyoto University, while the author was a student of the master and doctor courses of Kyoto University. Reports [7][48][50][80][81] are the results of the work with Prof. S. Muroga of University of Illinois, while the author was a research associate of University of Illinois.

Chapter 2. Theory of Completely Monotonic (Threshold) Functions

2.1. Introduction

This chapter investigetes fundamental properties of completely monotonic functions as a preparation for the network synthesis procedure given in the next chapter. A reason to deal with completely monotonic functions instead of threshold functions, which are the class of functions we want to use in the network design, is the difficulty in handling the asummability property derived from the theory of linear inequalities. This is compared with the 2-asummability property of completely monotonic functions, which can be handled directly by means of conventional Boolean expression, as will be shown.

The complete monotonicity was found to be a necessary condition for 1-realizability (realizability by a single threshold gate) by Paull and McCluskey[84], Muroga et al.[72], and Winder[98]. Moreover, this is known to be a considerably strict condition and, in fact, is also a sufficient condition for 1-realizability when completely specified functions of at most 8 variables are concerned [100][79].

Although complete monotonicity was at first defined for completely specified functions, we redefine it by 2-asummability, which was shown by Elgot[22] to be equivalent to complete monotonicity in the case of

completely specified functions, in order to treat both completely and incompletely specified functions in a unified form. A new concept, mutual monotonicity, is also introduced, resulting in a simple necessary and sufficient condition of complete monotonicity.
This mutual monotonicity can be easily examined for both completely and incompletely specified functions. Therefore, this method is frequently simpler than conventional testing methods of complete monotonicity, such as the ones devised by Muroga et al.,[72] and Winder[98].

The concept of mutual monotonicity also leads to a necessary and sufficient condition of complete monotonicity by means of direct comparisons of prime implicants. This is interesting because, though functional form of threshold functions has been investigated for some time, only necessary conditions or only sufficient conditions, but not both, are known[71][74][22], except in some special cases.

Finally, an extension of the above-mentioned concepts to threshold functions is also considered, emphasizing the importance of the isobaric (simultaneously realizable) condition.

2.2 Notations

We consider a logic function of n variables as a mapping from the set X^n of 2^n Boolean vector

$$x = (x_1,\ldots,x_i,\ldots,x_n) \quad x_i \in \{1, 0\} = X$$

to S= {1,0,* }:

$$f: X^n \to S. \tag{2.1}$$

Let,

$$\begin{aligned} U(f) &= f^{-1}(1) \\ V(f) &= f^{-1}(0) \\ W(f) &= f^{-1}(*). \end{aligned} \tag{2.2}$$

W(f) is called the set of redundant input vectors, or f is said to be defined on the set $U(f) \cup V(f)$, the domain of f. If $W(f) = \phi$, the null set, f, is a completely specified function, otherwise f is an incompletely specified function. When f and g have the same domain and assume the same value on it, they are regarded as the same functions.

For a subvector a, let N(a) represent the dimension of a. When $N(a) = k(\leq n)$, we define

$$f_a: X^{n-k} \to S$$

as follows:

$$\begin{aligned} U(f_a) &= \{y \mid (a, y) \in U(f)\} \\ V(f_a) &= \{y \mid (a, y) \in V(f)\} \\ W(f_a) &= \{y \mid (a, y) \in W(f)\}, \end{aligned} \tag{2.3}$$

where $y \in X^{n-k}$ and subvector a are collected to the left

by a certain permutation of variables for notational simplicity. A subvector $\mathbf{a}$ is frequently abbreviated as follows:

$$\boldsymbol{a} = (\overset{x_1}{1} - \overset{x_3}{0} - - \overset{x_6}{1}) \Rightarrow \boldsymbol{a} = 1\bar{3}6.$$

Negation of $\mathbf{a}$ is $\bar{\mathbf{a}}$; in the above example, $\bar{\mathbf{a}} = \bar{1}3\bar{6}$.

The following discussions are mostly true not only for completely specified functions but also for incompletely specified ones. However, if it is necessary to limit to completely specified functions, they are denoted by boldface types, like $\mathbf{f}$.

2.3. Threshold Functions

As is well known, a "threshold function" is a function f which has a weight vector W = $(w_1,\ldots,w_n)$, each component of which is a real number, and a real number T, called a threshold, such that

$$wx \geq T \quad \text{for } x \in U(f) \tag{2.4}$$

$$wx < T \quad \text{for } x \in V(f), \tag{2.5}$$

where WX is an inner product. If f is a threshold function it is said to be "1-realizable". When 2k vectors, not necessarily distinct, satisfy

$$\begin{aligned} &u_1, u_2, \cdots, u_k \in U(f) \\ &v_1, v_2, \cdots, v_k \in V(f) \\ u_1 + u_2 + \cdots + u_k &= v_1 + \cdots + v_k, \end{aligned} \tag{2.6}$$

f is called "k-summable", where + is the componentwise addition of vectors. If f is not k-summable for any possible combination of input vectors, it is "k-asummable". A function which is k-asummable for any k is a threshold function, and vice versa[13][22][27].

2.4. Completely Monotonic Functions

According to Muroga et al.[72] and Winder[98], completely monotonic functions are defined for completely specified functions as follows: a function is completely monotonic if and only if for any subvector a, $U(f_a) \geq U(f_{\bar{a}})$ or $U(f_{\bar{a}}) \geq U(f_a)$ holds, i.e., $U(f_a)$ and $U(f_{\bar{a}})$ are "comparable". Elgot[22] proved that this is equivalent to 2-asummability. We redefine complete monotonicity so as to include incompletely specified functions as well.

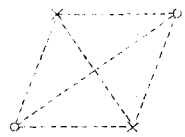
Definition 2.1: A function is "completely monotonic" if and only if it is 2-asummable.

2-summable vectors are geometrically shown in Fig. 2.1. Another implication is that if we arbitrarily choose two linear inequalities, one from each system of linear inequalities (2.4) and (2.5), and eliminate variable T from them, then we have a linear inequality containing only weights as variables. If we continue this procedure a system of linear inequalities which contain only weight variables will be obtained. The 2-summability is also equivalent to the existence of inconsistent inequalities in the resulting system of linear inequalities such as

$$\sum_{k_1} w_i > \sum_{k_2} w_j$$
$$\sum_{k_2} w_j > \sum_{k_1} w_i. \qquad (2.7)$$

The 2-asummability implies that these contradictions do not occur and, therefore, it is a necessary condition for 1-realizability.

Next, we introduce a binary relation between



o ∈ U(f)
× ∈ V(f)

Fig.2.1. Geometrical position of
2-summable vectors

subvectors a and $\bar{a}$. This is the same concept as the relation between input vectors defined by Muroga et al.[72], when completely specified functions are concerned.

Definition 2.2: Consider a function f. If for subvectors a and $\bar{a}$ there is subvector b, possibly N(b) = 0,such as

$$(a, b) \in U(f), \qquad (\bar{a}, b) \in V(f),$$

then, we define $a >^f \bar{a}$. If neither $a >^f \bar{a}$ nor $\bar{a} >^f a$ holds, it is denoted by $a =^f \bar{a}$. If both $a >^f \bar{a}$ and $\bar{a} >^f a$ hold, it is denoted by $a \diamond^f \bar{a}$.

The preceding concepts are unified as follows.

Theorem 2.1: The next three conditions are equivalent.

(a) f is completely monotonic.

(b) For f, there is no subvector a, N(a)>0, such that $a \diamond^f \bar{a}$.

(c) Let for a subvector a, N(a)>0,

$$(U(f_a) \cup V(f_a)) \cap (U(f_{\bar{a}}) \cup V(f_{\bar{a}})) = A_a,$$

then $U(f_a) \cap A_a$ and $U(f_{\bar{a}}) \cap A_a$ are comparable for any a.

Before proving this theorem we show the next lemma.

Lemma 2.1: If $u_1+u_2=v_1+v_2$ holds for four input vectors U_1, U_2, V_1, V_2, they can be written, by applying a certain permutation of variables, as

$$u_1 = (a, b, c)$$
$$v_1 = (\bar{a}, b, c)$$
$$u_2 = (\bar{a}, \bar{b}, c)$$
$$v_2 = (a, \bar{b}, c).$$

The converse is also true. Moreover, when the four vectors are all distinct, N(a)>0 and N(b)>0 hold.

Proof : Let a set of components of vectors where the four vectors take the same value be c, and in the remainder, let a set where U_1 and V_2 take the same value be a and a set where U_1 and V_1 take the same value be b ; then, obviously, four vectors are represented as in the lemma because of 2-summability. the converse is also clear. When four vectors are all distinct, clearly N(a)>0, N(b)>0. Q.E.D.

Proof of Theorem 2.1:

(a) $\Rightarrow$ (b). The proof is by contradiction. If $a <>^f \bar{a}$ holds for some subvector a, it is implied from Definition 2.2 that there are

$$u_1 = (a, b) \in U(f)$$
$$v_1 = (\bar{a}, b) \in V(f)$$
$$u_2 = (\bar{a}, c) \in U(f)$$
$$v_2 = (a, c) \in V(f).$$

For these vectors, obviously,

$$u_1 + u_2 = (a + \bar{a}, b + c) = v_1 + v_2$$

holds and f is 2-summable. Hence a contradistion.

(b) $\Rightarrow$ (a). Assume f is not completely monotonic. There are four vectors

$$u_1, u_2 \in U(f), \quad v_1, v_2 \in V(f)$$
$$u_1 + u_2 = v_1 + v_2, \tag{2.8}$$

since f is 2-summable by Definition 2.1. Moreover, the same vector cannot belong to both U(f) and V(f) simultaneously, and if $u_1 = u_2$ in (2.8) then $u_1 = u_2 = v_1 = v_2$, since these vectors are all Boolean. Hence, the four vectors can be assumed to be all distinct. Therefore, from Lemma 2.1,

$$u_1 = (a, b, c) \qquad u_2 = (\bar{a}, \bar{b}, c)$$
$$v_1 = (\bar{a}, b, c) \qquad v_2 = (a, \bar{b}, c),$$

where N(a)>0 and N(b)>0. Comparisons of u_1 and v_1 and u_2 v_2 yield $a >^f \bar{a}$ and $\bar{a} >^f a$, respectively. Thus, we have $a \diamond^f \bar{a}$ and this is a contradiction.

(a) $\Rightarrow$ (c). Assume $U(f_{\bar{a}}) \cap A_a$ and $U(f_a) \cap A_a$ are incomparable. Then there are vectors r and s such that

$$r \in U(f_a) \cap A_a$$
$$r \notin U(f_{\bar{a}}) \cap A_a$$
$$s \notin U(f_a) \cap A_a$$
$$s \in U(f_{\bar{a}}) \cap A_a.$$

However, $r \notin U(f^a) \cap A_a$ implies $r \in V(f_{\bar{a}}) \cap A_a$ since r belongs to A_a. Therefore,

$$(a, r) \in U(f)$$
$$(\bar{a}, r) \in V(f)$$
$$(a, s) \in V(f)$$
$$(\bar{a}, s) \in U(f),$$

and this implies f is not completely monotonic from Definition 2.1.

(c) $\Rightarrow$ (a). This is easily obtained by reversing the previous proof. Q.E.D.

Condition (c) of Theorem 2.1 also shows that Definition 2.1 is equivalent to that of Muroga et al.[72] and Winder[98] when completely specified functions are considered, as first proved by Elgot[22].

Example 2.1: Consider a function f of three variables

$$\begin{aligned} U(f) &= \{101, 110, 010\}, \\ V(f) &= \{111, 000\}, \\ W(f) &= X^3 - (U(f) \cup V(f)). \end{aligned} \tag{2.9}$$

Compare (101) $\in$ U(f) and (111) $\in$ V(f). In this case a = (-0-) = $\bar{2}$ and b = (1-1) = 13. From Definition 2.2, we have a>$\bar{a}$, namely $\bar{2} >^{f} 2$. Similarly,

$$\begin{aligned} &(101) \text{ and } (000) \Rightarrow 13 >^{f} \bar{1}\,\bar{3}, \\ &(110) \text{ and } (111) \Rightarrow \bar{3} >^{f} \dot{3}, \\ &(110) \text{ and } (000) \Rightarrow 12 >^{f} \bar{1}\,\bar{2}, \\ &(010) \text{ and } (111) \Rightarrow \bar{1}\,\bar{3} >^{f} 13, \\ &(010) \text{ and } (000) \Rightarrow 2 >^{f} \bar{2}. \end{aligned} \tag{2.10}$$

For all other subvectors a, $a =^{f} \bar{a}$. From these results, $2 <>^{f} \bar{2}$ and $13 <>^{f} \bar{13}$ are obtained and, therefore, f is not completely monotonic by Theorem 2.1. Indeed, (101) + (010) = (111) + (000) and f is 2-summable.

Here we note the fact that there exist incompletely specified functions which can not be extended to completely specified functions, preserving complete monotonicity, by adding each redundant vector to either U(f) or V(f) in any way, despite the fact that incompletely specified threshold functions can always be extended to completely specified threshold functions. An example is the following function of three variables.

Example 2.2: Consider a function such that (see Fig.2.2)

$$U(f) = \{100, 111, 010\}$$
$$V(f) = \{110, 001\}. \qquad (2.11)$$

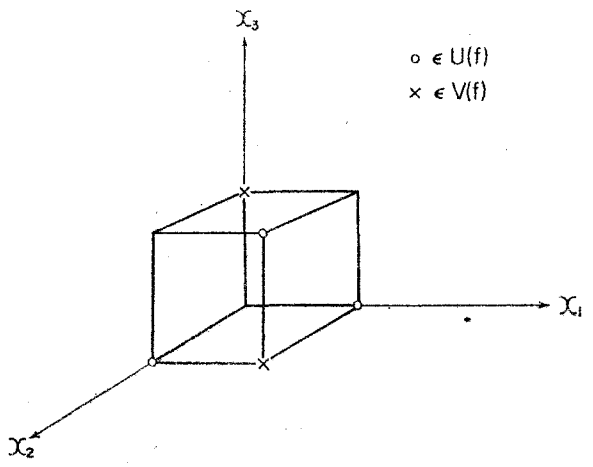


Fig. 2.2. A function which cannot be extended to be completely specified while preserving complete monotonicity.

This function is completely monotonic from Definition 2.1. If we add one of the redundant vectors, (011), (101) and (000), to either U(f) or V(f), however, the resulting function is no longer completely monotonic.

A necessary and sufficient condition for a function having the above property was investigated by Kambayashi, and it is to satisfy the following two conditions.

(1) There are vectors such that

$$\begin{aligned} &\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3 \in U(f) \\ &\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3 \in V(f) \\ &\boldsymbol{u}_1 + \boldsymbol{u}_2 + \boldsymbol{u}_3 = \boldsymbol{v}_1 + \boldsymbol{v}_2 + \boldsymbol{v}_3, \end{aligned} \tag{2.12}$$

where vectors are not necessarily distinct.

(2) For some i, j, k($1 \leq i,j,k \leq 3$),

$$(u_i + u_j)_l = (v_k)_l + \begin{cases} 1 \\ 0 \end{cases}$$

or (2.13)

$$(v_i + v_j)_l = (u_k)_l + \begin{cases} 1 \\ 0 \end{cases}, \quad l = 1, 2, \cdots, n$$

holds.

However, we do not discuss this property further.

2.5. Mutual Monotonicity

In this section we introduce a new concept, mutual monotonicity, to facilitate the test of complete monotonicity.

Definition 2.3: For functions g and h of the same variables, if there is no set of vectors such that

$$\begin{aligned} u_g &\in U(g), \qquad u_h \in U(h) \\ v_g &\in V(g), \qquad v_h \in V(h) \end{aligned} \tag{2.14}$$

and

$$u_g + u_h = v_g + v_h,$$

then g and h are called "mutually monotonic"* and denoted as $g \sim h$. If g and h are not mutually monotonic, they are denoted as $g \not\sim h$.

This definition does not require the complete monotonicity of g and h themselves. Obviously, this binary relation of functions is symmetric, i.e., $g \sim h \Rightarrow h \sim g$, but it is neither reflective nor transitive.

This concept can be expressed in other ways.

Theorem 2.2: For functions g and h of the same variables, the following three conditions are equivalent.

(a) g and h are mutually monotonic.

(b) There is no subvector $\mathbf{a}$, $N(\mathbf{a})>0$, such that $\mathbf{a} >^{g} \bar{\mathbf{a}}$ and $\bar{\mathbf{a}} >^{h} \mathbf{a}$ hold simultaneously.

---

*Strictly apeaking, this relation is said to be completely mutually monotonic in analogy with complete monotonicity. Since there is no confusion, we use this terminology in this paper.

(c) Let

$$(U(g_a) \cup V(g_a)) \cap (U(h_{\bar{a}}) \cup V(h_{\bar{a}})) = B_a.$$

$U(g_a) \cap B_a$ and $U(h_{\bar{a}}) \cap B_a$ are comparable for any subvector a, possibly N(a) = 0.

Proof:

(a) $\Rightarrow$ (b). Assume that condition (b) does not hold. Then we have the following vectors:

$$\begin{aligned} u_g &= (a, b) \in U(g), & u_h &\in (\bar{a}, c) \in U(h) \\ v_g &= (\bar{a}, b) \in V(g), & v_h &\in (a, c) \in V(h). \end{aligned} \tag{2.15}$$

Obviously,

$$u_g + u_h = v_g + v_h,$$

and hence g $\not\sim$ h, a contradiction.

(b) $\Rightarrow$ (c). Assume that condition (c) does not hold. Then for some a, $U(g_a) \cap B_a$ and $U(h_{\bar{a}}) \cap B_a$ are not comparable. This implies the existence of vectors such that

$$\begin{aligned} (a, b) &= (a, d, e) \in U(g) \\ (\bar{a}, b) &= (\bar{a}, d, e) \in V(h) \\ (a, c) &= (a, d, \bar{e}) \in V(g) \\ (\bar{a}, c) &= (\bar{a}, d, \bar{e}) \in U(h). \end{aligned} \tag{2.16}$$

N(e)>0 because b $\neq$ c must hold. Thus, we have $e >^{g} \bar{e}$ and $\bar{e} >^{h} e$, and this is a contradiction.

(c) $\Rightarrow$ (a). Assume that g and h are not mutually monotonic. Then there are four vectors as shown in (2.14). These vectors may be written as follows, according to Lemma 2.1:

$$u_g = (a, b, c), \qquad u_h = (\bar{a}, \bar{b}, c)$$
$$v_g = (a, \bar{b}, c), \qquad v_h = (\bar{a}, b, c).$$

Then $U(g_a) \cap B_a$ and $U(h_{\bar{a}}) \cap B_a$ are not comparable because

$$(b, c) \in U(g_a) \cap B_a, \qquad \notin U(h_{\bar{a}}) \cap B_a$$
$$(\bar{b}, c) \notin U(g_a) \cap B_a, \qquad \in U(h_{\bar{a}}) \cap B_a.$$

Q.E.D.

Corollary 2.2A: A constant function is mutually monotonic with any function. The converse is also true.

Corollary 2.2B: When g and h are completely specified, condition (c) becomes as follows. For any subvector a, possibly $N(a) = 0$, $U(g_a)$ and $U(h_{\bar{a}})$ are comparable.

Mutual monotonicity is related to complete monotonicity in the following theorem.

Theorem 2.3: The following four conditions are equivalent.

(a) f is completely monotonic.

(b) $f \sim f$.

(c) For any i ($1 \leq i \leq n$), $f_i \sim f_{\bar{i}}$*.

(d) For a given i, f_i and $f_{\bar{i}}$ are both completely monotonic and $f_i \sim f_{\bar{i}}$.

---

*f_i and $f_{\bar{i}}$ may be expressed as

$$f = f_i x_i \vee f_{\bar{i}} \bar{x}_i .$$

This expansion is possible even when f is incompletely specified. In general, f_i and $f_{\bar{i}}$ are also incompletely specified functions.

Proof: (a) $\Leftrightarrow$ (b) may be trivial from the definition. We will show (a) $\Rightarrow$ (c) $\Rightarrow$ (d) $\Rightarrow$ (a). Proofs are all by contradiction.

(a) $\Rightarrow$ (c). Assume $f_i \propto f_{\bar{i}}$ for some i, then by Definition 2.3, there are vectors such that

$$u_1 \in U(f_i), \quad u_2 \in U(f_i)$$
$$v_1 \in V(f_i), \quad v_2 \in V(f_i)$$
$$u_1 + u_2 = v_1 + v_2.$$

Attach to each vector a component corresponding to x_i and write it to the left side, then

$$(1, u_1), (0, u_2) \in U(f)$$
$$(1, v_1), (0, v_2) \in V(f)$$
$$(1, u_1) + (0, u_2) = (1, v_1) + (0, v_2).$$

Thus f is not completely monotonic.

(c) $\Rightarrow$ (d). To prove this it is sufficient to show that f_i and $f_{\bar{i}}$ are both completely monotonic for the given i. Assume f_i (in the case of $f_{\bar{i}}$ the proof is similar) is not completely monotonic, i.e., there are distinct vectors

$$u_1, u_2 \in U(f_i)$$
$$v_1, v_2 \in V(f_i)$$
$$u_1 + u_2 = v_1 + v_2.$$

From Lemma 2.1 and attaching the i-th component to the left, we have

$$(1, u_1) = (1, a, b, c)$$
$$(1, v_1) = (1, \bar{a}, b, c)$$
$$(1, u_2) = (1, \bar{a}, \bar{b}, c)$$
$$(1, v_2) = (1, a, \bar{b}, c),$$

where $N(a)>0$ and $N(b)>0$. Then, without loss of generality, pick up the j-th component which is included in a and suppose that $a_j = 1$. Remove the j-th component from a and $\bar{a}$, and let the resulting subvectors be a' and $\bar{a}'$, respectively. Then

$$s_1 = (1, a', b, c) \in U(f_j)$$
$$t_1 = (1, a', \bar{b}, c) \in V(f_j)$$
$$s_2 = (1, \bar{a}', \bar{b}, c) \in U(f_j)$$
$$t_2 = (1, \bar{a}', b, c) \in V(f_j),$$

and, moreover,

$$s_1 + s_2 = t_1 + t_2,$$

which implies $f_j \sim f_j$. Hence a contradiction. By this argument we have proved that if condition (b) holds then f_i and $f_{\bar{i}}$ are completely monotonic and $f_i \sim f_{\bar{i}}$ for any i.

(d) $\Rightarrow$ (a). Assume f is not completely monotonic. Then, by Definition 2.1 and Lemma 2.1, there are four vectors such that

$$(a, b, c), (\bar{a}, \bar{b}, c) \in U(f)$$
$$(\bar{a}, b, c), (a, b, \bar{c}) \in V(f). \qquad (2.17)$$

Suppose first that the i-th component is included in c and that $c_i = 1$ without loss of generality. Then (2.17) implies

$$(\boldsymbol{a}, \boldsymbol{b}, c'), (\bar{\boldsymbol{a}}, \bar{\boldsymbol{b}}, c') \in U(f_i)$$
$$(\bar{\boldsymbol{a}}, \boldsymbol{b}, c'), (\boldsymbol{a}, \bar{\boldsymbol{b}}, c') \in V(f_i), \qquad (2.18)$$

which is equivalent to saying that f_i is not completely monotonic since f_i is 2-summable. Suppose next that the i-th component is included in $\mathbf{a}$ and also that $a_i = 1$. Then (2.17) implies

$$(\boldsymbol{a}', \boldsymbol{b}, c) \in U(f_i)$$
$$(\boldsymbol{a}', \bar{\boldsymbol{b}}, c) \in V(f_i)$$
$$(\bar{\boldsymbol{a}}', \bar{\boldsymbol{b}}, c) \in U(f_i)$$
$$(\bar{\boldsymbol{a}}', \boldsymbol{b}, c) \in V(f_i),$$

and this implies $f_i \not\propto f_{\bar{i}}$, where c' and a' are c and a, respectively, from which the i-th component was removed. When the i-th component is included in $\mathbf{b}$, we have also $f_i \not\propto f_{\bar{i}}$ by similar argument. Consequently, we have found that if condition (a) does not hold, then either $f_i \not\propto f_{\bar{i}}$ holds or f_i (or $f_{\bar{i}}$) is not completely monotonic, resulting in a contradiction. Q.E.D.

2.6 Testing Method for Complete Monotonicity

By the argument in the previous section, we have obtained three testing methods for complete monotonicity by means of mutual monotonicity.

(a) By Theorem 2.3, condition (b); examine if $f \sim f$.

(b) By Theorem 2.3, condition (c); examine if $f_i \sim f_{\bar{i}}$ for $i = 1,2,\ldots,n$.

(c) By Theorem 2.3, condition (d); examine if $f_i \sim f_{\bar{i}}$ for some i. If $f_i \sim f_{\bar{i}}$, then expand f_i and $f_{\bar{i}}$, respectively, to determine their complete monotoicity. If we continue the expansion of obtained functions successively, we will eventually have functions of one variable. Any function of one variable is known to be completely monotonic. Therefore, if all pairs of functions obtained from this process of expansion are mutually monotonic, f is shown to be completely monotonic by successive applications of Theorem 2.3, condition (d). On the other hand, if we have at least one pair of functions which is not mutually monotonic in the process of expansion, f is not completely monotonic.

Hereafter, we will confine ourselves to method (c). First, because it is desirable to examine the mutual monotonicity of functions of few variables, since the testing of mutual monotonicity becomes progressively difficult with the increase of the number of variables, as will be shown later; and second, because method (c) yields a clue to compound synthesis even if f itself is

not completely monotonic .

To determine complete monotonicity by method (c), we have to accomplish a function tree as shown in Fig. 2.3.

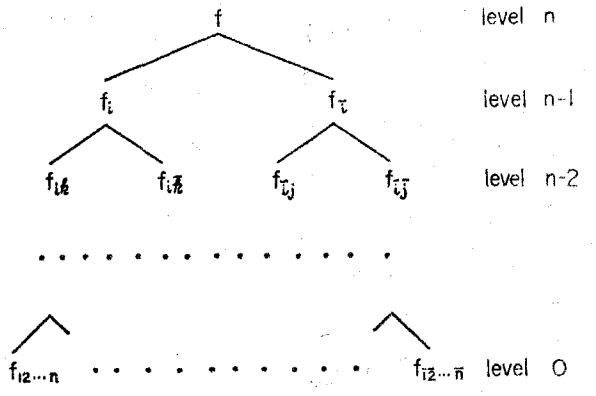


Fig. 2.3. Expansion diagram of a function.

For a function of n variables, each level is denoted as level n, level n-1,..., level 0 from the top. Generally, the order of variables in the expansion is arbitrary. However, all functions in a level are usually expanded by the same variable unless there is some special requirement. We call the function tree an "expansion diagram" of f if the relation of mutual monotonicity of functions is written on the diagram.

The process of constructing the expansion diagram in order to examine complete monotonicity of a certain function f can be partly omitted if obtained functions are known to be either completely monotonic or not. If an obtained function is completely monotonic, the expansion of that function yields only mutually monotonic pairs of functions. If an obtained function is not completely

monotonic, the original function f itself is not completely monotonic.

The mutual monotonicity of each pair of functions may be examined in several ways, among them the direct examination of vectors based on Definition 2.3. This procedure is simple but may need a vast amount of comparisons of vectors and, therefore, is somewhat unrealistic except for sparsely specified functions.

The examination of condition (b) of Theorem 2.2 is essentially the same as the above method.

Condition (c) of Theorem 2.2 is interesting because the method based on this would be quite similar to the method of examining complete monotonicity proposed by Muroga et al.[72] and Winder[98]. By incorporating conceivable simplifications which may be discussed in conjunction with the method for complete monotonicity, this method could be a practical one. In this paper, however, we do not discuss these methods any further. We present a new method in Section 2.9 which consists of the comparison of product terms of given functions, since this seems, not always, but often simpler than other methods and may reveal a new aspect of completely monotonic functions.

2.7 Use of Symmetric Variables

Before stating the method of examining mutual monotonicity by algebraic manipulation, we deal with two possibilities which facilitate testing of complete monotonicity in the following sections.

Frequently, a function is symmetric in more than one variable; in that case, part of the determination of mutual monotonicity can be eliminated by the following theorem.

Theorem 2.4: Suppose a function f in level j is symmetric in m variables ($m \leq j$) and f is expanded successively by the symmetric variables. Then if two functions in level j-1 which are obtained from f are mutually monotonic, all pairs of functions in level j-2 through level j-m obtained from f are also mutually monotonic.

Proof: Without loss of generality, suppose that f is symmetric in $\{x_1, x_2, \ldots, x_m\}$ and f is expanded by x_1, x_2, ..., x_m, successively. From the assumption in the theorem, $f_1 \sim f_{\bar{1}}$. Then, assume that we have a non-mutually monotonic pair of functions $f_{ak} \propto f_{\bar{a}k}$ when functions are expanded by x_k ($1<k\leq m$). From the symmetry of variables, this implies $f_{b1} \propto f_{b\bar{1}}$ for some b. However, $f_{b1} \propto f_{b\bar{1}}$ implies $f_1 \propto f_{\bar{1}}$ from Definition 2.3. Thus a contradiction.

Q.E.D.

By this theorem, we can omit the test of pairs of functions in level j-2 through level j-m, if we know the mutual monotonicity of functions in level j-1 obtained from a function in level j which is symmetric in m variables.

2.8 Dual Functions

In threshold logic, dual functions are known to play an interesting role[37]. They exhibit interesting properties in our theory as well. In order to also deal with incompletely specified functions, the dual function is defined as follows.

Definition 2.4: "Dual function" f^d of a function f is defined by

$$\begin{aligned} U(f^d) &= \{\bar{x} \mid x \in V(f)\} \\ V(f^d) &= \{\bar{x} \mid x \in U(f)\} \\ W(f^d) &= \{\bar{x} \mid x \in W(f)\}. \end{aligned} \tag{2.19}$$

Clearly, $(f^d)^d = f$, and $f = g$ implies $f^d = g^d$, and vice versa.

An interesting relationship of an original function and the dual function is summarized in the next lemma.

Lemma 2.2: The binary relations of Definition 3.2 for f, $>^f$, $=^f$, $<>^f$, correspond to $>^{f^d}$, $=^{f^d}$, $<>^{f^d}$, respectively, for f^d.

Proof: Suppose $a >^f \bar{a}$, namely, there are two input vectors

$$\begin{aligned} &(a, b) \in U(f) \\ &(\bar{a}, b) \in V(f), \end{aligned}$$

then we have from Definition 2.4

$$\begin{aligned} &(a, \bar{b}) \in U(f^d) \\ &(\bar{a}, \bar{b}) \in V(f^d). \end{aligned}$$

This implies $a >^{f^d} \bar{a}$. Q.E.D.

However, the converse of Lemma 2.2 (i.e., if $>^f$, $=^f$, $<>^f$ and $>^g$, $=^g$, $<>^g$, respectively, correspond, then $g = f^d$), does not hold, in general; it holds when f is a completely specified, completely monotonic function.

The following two theorems are immediate consequences of Lemma 2.2, Theorem 2.1, condition (b), and Theorem 2.2, condition (b).

Theorem 2.5: If f is completely monotonic then f^d is also completely monotonic, and vice versa.

Theorem 2.6: If $g \vee h$, then $g^d \vee h$.

Therefore, we can replace any function obtained in the expansion diagram by its dual if only the complete monotonicity of the given function is concerned. Note that, however, if there are non-mutually monotonic pairs in the expansion diagram, their positions may change by the replacement of functions by their duals, although their total number remains the same.

2.9 Test of Mutual Monotonicity by Algebraic Methods

In this section we deal with a method of determining mutual monotonicity by algebraic manipulations. This method can be easily applied by hand computation to functions of up to about 7 ∿ 8 variables, or more if they are symmetric in some variables or have certain particular properties. Three theorems are given for general functions, completely specified functions, and completely specified unate functions, respectively. They are based essentially on the same idea. The determination of mutual monotonicity by each method becomes progressively easier though the application would be more restricted.

For simplicity, let a function be expressed by a sum-of-products form. A produt term, is represented by a, b, c, etc., with or without a suffix. Sometimes a product term is factored into sets of literals and written as $a = a_1a_2a_3$, and a is said to be "partitioned" into $a_1a_2a_3$. In this case no two subproduct terms such as a_i and a_j involve common literals. Let the subvector corresponding to a be $\mathbf{a}$. For example, $\mathbf{a}$ = (1-01) corresponds to $a = x_1\bar{x}_3x_4$. $a \to b$ implies that a subsumes* b, and N(a) is the number of literals which appear in a. $\bar{a}$ is the negation of a, i.e., the product term obtained by negating all the literals in a.

---

*A product term a "subsumes" another product term b if all the literals in b are also in a. For example, $x_1\bar{x}_2x_4 \to \bar{x}_2x_4$.

A. General Case

Let f be an incompletely or completely specified function. In order to treat an incompletely specified function by the algebraic expression, two completely specified functions f^1 and f^0 are associated with the function f, where

$$U(f^1) = U(f), \quad V(f^1) = V(f) \cup W(f)$$
$$U(f^0) = V(f), \quad V(f^0) = U(f) \cup W(f). \qquad (2.20)$$

Obviously, the function f is completely described by f^1 and f^0. Note that when f is completely specified, $f^1 = f$ and $f^0 = \bar{f}$.

Theorem 2.7: A necessary and sufficient condition for g and h to be mutually monotonic is that when g^1 and h^1 are expressed by sum-of-products forms, there are no product terms $a_1a_2b_1b_2c$ and $a_3a_4\bar{b}_1\bar{b}_2c$ such that

$$g^1 = a_1a_2b_1b_2c \vee G$$

$$U(\bar{a}_1a_2a_3\bar{b}_1b_2c) \cap U(g^0) \neq \phi ,$$

and

$$h^1 = a_3a_4\bar{b}_1\bar{b}_2c \vee H$$

$$U(a_1\bar{a}_3a_4b_1\bar{b}_2c) \cap U(h^0) \neq \phi ,$$

and

$$N(a_1) + N(b_1) > 0 \quad \text{and} \quad N(a_3) + N(b_1) > 0,$$

where G and H represent the remainder of g^1 and h^1, respectively, and there is no common variable in a_1a_2

and a_3a_4.

Proof: Proof is by contradiction.

Sufficiency: Assume g $\not\sim$ h. Then by Theorem 2.2, condition (b) there are four vectors such that

$$\left.\begin{aligned}(r, s) &\in U(g)\\ (\bar{r}, s) &\in V(g)\end{aligned}\right\}, \tag{2.21}$$

$$\left.\begin{aligned}(\bar{r}, t) &\in U(h)\\ (r, t) &\in V(h)\end{aligned}\right\}, \tag{2.22}$$

$$N(r) > 0.$$

From (2.21), there is a product term r_1s_1 in g^1 where $r \to r_1$, $s \to s_1$, and $N(r_1)>0$. For $N(r_1) = 0$, $(r,s) \in U(g)$ implies $(\bar{r},s) \in U(g)$, and this is a contradiction. Similarly, there is a product term r_2s_2 in h^1 where $r \to r_2$, $t \to s_2$ and $N(r_2)>0$. It is possible to rewrite r_1 and r_2 as follows

$$r_1 = r_1'r_3', \qquad \bar{r}_2 = \bar{r}_2'\bar{r}_3'.$$

In other words, r_1' and r_2' have no common variable.

Next, consider common variables in s_1 and s_2 and collect separately those of the same sign and those of the opposite sign, and let them be s_3' and s_4', respectively. Then,

$$s_1 = s_1's_3's_4', \qquad s_2 = s_2's_3'\bar{s}_4',$$

where there is no common variable in s_1' and s_2'. Consequently, there are product terms

$$r_1'r_3's_1's_3's_4' \quad \text{and} \quad \bar{r}_2'\bar{r}_3's_2's_3'\bar{s}_4'$$

in g^1 and h^1, respectively. Moreover, (2.21) implies

$U(\bar{r}s) \cap U(g^0) \neq \phi$ and thus

$$U(\bar{r}_1'\bar{r}_2'\bar{r}_3's_1's_3's_4') \cap U(g^0) \neq \emptyset,$$

because $\bar{r}s \to \bar{r}_1'\bar{r}_2'\bar{r}_3's_1's_3's_4'$. Similarly,

$$U(r_1'r_2'r_3's_2's_3'\bar{s}_4') \cap U(h^0) \neq \emptyset.$$

Then if we put $r_1' = a_1$, $s_1' = a_2$, $r_3' = b_1$, $s_4' = b_2$, $s_3' = c$, $\bar{r}_2' = a_3$ and $s_2' = a_4$, we have proved the sufficiency of the theorem. Clearly, $N(r_1)>0$ implies $N(a_1) + N(b_1)>0$ and $N(r_2)>0$ implies $N(a_3) + N(b_1)>0$.

Necessity: Assume there are product terms that satisfy the conditions in the theorem. Then there are four vectors such that

$$(a_1, a_2, *_1, *_2, b_1, b_2, c, *_3) \in U(g^1)$$

$$(\bar{a}_1, a_2, a_3, \square_1, \bar{b}_1, b_2, c, \square_2) \in U(g^0)$$

$$(*_4, *_5, a_3, a_4, \bar{b}_1, b_2, c, *_6) \subset U(h^1)$$

$$(a_1, \square_3, \bar{a}_3, a_4, b_1, \bar{b}_2, c, \square_4) \in U(h^0),$$

where $*$ implies any subvector and $\square$ implies some subvector. Thus, if we choose the $*$ part appropriately, we get

$$\begin{aligned} a_1\bar{a}_3b_1 &\overset{g}{>} \bar{a}_1a_3\bar{b}_1 \\ a_1\bar{a}_3b_1 &\overset{h}{<} \bar{a}_1a_3\bar{b}_1, \end{aligned} \qquad (2.23)$$

and obviously $N(a_1) + N(a_3) + N(b_1)>0$. This implies $g \not\propto h$. Thus a contradiction. Q.E.D.

It is possible to eliminate the condition $N(a_1) + N(b_1) > 0$ and $N(a_3) + N(b_1) > 0$ from Theorem 2.7 because if, for example, $N(a_1) + N(b_1) = 0$, we automatically have

$$U(\bar{a}_1 a_2 a_3 \bar{b}_1 b_2 c) = U(a_2 a_3 b_2 c) \subseteq U(a_2 b_2 c)$$
$$= U(a_1 a_2 b_1 b_2 c) \subseteq U(g^1)$$

and this implies

$$U(\bar{a}_1 a_2 a_3 \bar{b}_1 b_2 c) \cap U(g^0) = \emptyset.$$

By this theorem, we can examine the mutual monotonicity of g and h as follows. First, represent g^1 and h^1 in sum-of-products forms and simplify them, if possible. Then, apply the following procedures to every pair of product terms selected from g^1 and h^1, respectively.

(1) Among variables which appear in both product terms, let the part of the same sign and that of the opposite sign be c and b, respectively; the remainder is denoted a and a' for g^1 and h^1, respectively. Namely, the pair of product terms of g^1 and h^1 is written as abc and $a'\bar{b}c$.

(2) Partition a, a', and b as $a = a_1 a_2$, $a' = a_3 a_4$, and $b = b_1 b_2$ under the conditions $N(a_1) + N(b_1) > 0$, $N(a_3) + N(b_1) > 0$. For every partition, examine if both

$$U(\bar{a}_1 a_2 a_3 \bar{b}_1 b_2 c) \cap U(g^0) \neq \emptyset$$
$$U(a_1 \bar{a}_3 a_4 b_1 \bar{b}_2 c) \cap U(h^0) \neq \emptyset$$

are satisfied. If no pair of partition of product terms selected from g^1 and h^1 satisfies (2) then $g \sim h$, and if there is at least one pair which satisfies (2) then $g \not\sim h$.

Though this method seems to be very complicated, the insight into the structures of g and h sometimes greatly reduces the amount of computation. For example, some of the subproduct terms are frequently null, and if $N(a) + N(b) = 0$ or $N(a') + N(\bar{b}) = 0$ is known, it is not necessary to proceed to procedure (2). Considering the form of g^0 and h^0, try only partitions which may yield $U(\bar{a}_1 a_2 a_3 \bar{b}_1 b_2 c)$ and $U(a_1 \bar{a}_3 a_4 b_1 \bar{b}_2 c)$ that intersect $U(g^0)$ and $U(h^0)$ respectively. Other techniques will be found in the examples. When g and h are very sparsely specified, however, the procedure due to Definition 2.1 may sometimes be simpler than this procedure.

Example 2.5: Examine the mutual monotonicity of incompletely specified g and h:

g					h				
x_1	x_2	x_3	x_4	g	x_1	x_2	x_3	x_4	h
1	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	0	1
1	0	0	1	1	1	0	1	0	1
1	0	0	0	1	0	1	1	0	1
1	1	1	0	1	0	0	1	0	1
0	1	1	0	1	0	1	0	0	0
0	0	0	1	0	1	0	0	1	0
0	0	0	0	0	0	0	0	0	0

$$g: \boldsymbol{g}^1 = x_1\bar{x}_2 \vee x_2x_3\bar{x}_4, \quad \boldsymbol{g}^0 = \bar{x}_1\bar{x}_2\bar{x}_3$$
$$h: \boldsymbol{h}^1 = x_1\bar{x}_2x_3 \vee x_3\bar{x}_4, \quad \boldsymbol{h}^0 = \bar{x}_1\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_3x_4. \qquad (2.24)$$

Now compare $x_1\bar{x}_2$ and $x_1\bar{x}_2x_3$. In this pair, however, $N(a) + N(b) = 0$ holds and procedure (2) is omitted. Similarly for $x_2x_3\bar{x}_4$ and $x_3\bar{x}_4$. Next compare $x_1\bar{x}_2$ and $x_3\bar{x}_4$. By procedure (1) $a = x_1\bar{x}_2$ and $a' = x_3\bar{x}_4$. Considering that x_3 appears in every term of h^0, necessarily $a_3 = x_3$ or $a_3 = x_3\bar{x}_4$, i.e., a_3 contains x_3, for $U(a_1\bar{a}_3a_4b_1\bar{b}_2c)$ to intersect $U(h^0)$. This implies that x_3 necessarily appears in $\bar{a}_1a_2a_3\bar{b}_1b_2c$ which is compared with $U(g^0)$, and obviously from the form of g^0 the term does not intersect $U(g^0)$. Finally, for $x_2x_3\bar{x}_4$ and $x_1\bar{x}_2x_3$, $c = x_3$ and this implies that x_3 appears in $\bar{a}_1a_2a_3\bar{b}_1b_2c$ which is compared with $U(g^0)$, and thus the term does not intersect $U(g^0)$. Thus we have proved that $g \sim h$.

Completely Specified Functions

In this case, Theorem 2.7 is simplified to some extent as follows.

Theorem 2.8: A necessary and sufficient condition for two completely specified functions g and h to be mutually monotonic, when g and h are represented by sum of prime implicants*, is to satisfy the following two conditions.

(a) There is no variable which appears in g and h, respectively, with opposite signs**.

(b) There is no pair of prime implicants a_1a_2c and a_3a_4c such that

$$\begin{aligned} &g = a_1a_2c + G \\ &U(\bar{a}_1a_2a_4c) \cap V(g) \neq \emptyset, \\ &h = \bar{a}_3a_4c + H \\ &U(a_1\bar{a}_3a_4c) \cap V(h) \neq \emptyset, \end{aligned} \qquad (2.25)$$

and $N(a_1)>0$, $N(a_3)>0$, where G and H represent the remainder of g and h, respectively, and there is no common variable in a_1a_2 and a_3a_4.

Proof: We first prove that condition (a) is necessary. Assume that condition (a) does not hold, i.e., there are prime implicants x_iabc and $\bar{x}_ia'\bar{b}c$ in g and h, respectively. The existence of the prime implicant x_iabc in g implies the

---

*A product term a is a prime implicant of a function f if and only if $U(a) \subseteq U(f)$ and there is no product term b such that $U(b) \subseteq U(f)$ and $a \rightarrow b$.

**The sign of a negated (unnegated) variable is negative (positive).

existence of vectors

$$(1, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, *) \in U(g)$$
$$(0, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \square) \in V(g), \qquad (2.26)$$

where $*$ implies an arbitrary subvector and $\square$ implies a certain subvector as before. The i-th component is written to the left of the vectors. From (2.26), if we fix the $*$ part to be the same as the $\square$ part, $i >^{g} \bar{i}$ is obtained. Similarly, it is possible to obtain $\bar{i} >^{h} i$, and this implies $g \not\asymp h$. Therefore, we can suppose that condition (a) is satisfied. This is equivalent to saying that we suppose $N(b_1) = N(b_2) = 0$ in Theorem 2.7. Then, considering that $U(g^0) = V(g)$ when g is completely specified, etc., we can prove condition (b) is a necessary and sufficient condition for mutual monotonicity of g and h, under condition (a). Q.E.D.

By this theorem we can examine the mutual monotonicity of two completely specified functions g and h in the same manner as incompletely specified functions. First, represent g and h by sum of prime implicants. If there are variables which appear in g and h with opposite signs, we have $g \not\asymp h$. If there are no such variables, apply the following procedure to every pair of prime implicants, each selected from g and h, respectively.

(1) Let the literals which appear in both prime implicants be c and the remainder be a and a' for g and h, respectively. Namely, the pair of prime implicants of g and h is written as ac and

a'c, respectively.

(2) Partition a and a' as $a = a_1a_2$ and $a' = a_3a_4$ under constraints $N(a_1)>0$, $N(a_3)>0$, and for every partition examine if both

$$U(\tilde{a}_1a_2a_3c) \cap V(\boldsymbol{g}) \neq \varnothing$$

$$U(a_1\tilde{a}_3a_4c) \cap V(\boldsymbol{h}) \neq \varnothing$$

are satisfied. If no pair of partions of prime implicants selected from g and h satisfies (2), then $g \sim h$, and if at least one partition satisfies (2), then $g \nsim h$.

From Corollary 2.2B, it is obvious that $U(g) \supseteq U(h)$ or $U(h) \supseteq U(g)$ must hold for g and h to be mutually monotonic. Assume $U(g) \supseteq U(h)$. Then we can add a condition $N(a_4)>0$ to condition (b) of Theorem 2.8, because if $N(a_4) = 0$,

$$U(\tilde{a}_1a_2a_3c) \subseteq U(a_3c) = U(a'c) \subseteq U(\boldsymbol{h}) \subseteq U(\boldsymbol{g})$$

holds, and thus we immediately have

$$U(\tilde{a}_1a_2a_3c) \cap V(\boldsymbol{g}) = \varnothing.$$

By employing this property, we can eliminate the test of some partitions of a and a' in procedure (2). Procedure (2) is especially unnecessary if $N(a') \leq 1$ holds.

A wellknown necessary condition for complete monotonicity (as will also be shown in Theorem 2.10) is unateness*.

---

*A function is unate if and only if it has an expression in which each variable appears with negation or without negation throughout.

Suppose that some of g and h is not unate. If we are to examine the mutual monotonicity of g and h in order to examine the complete monotonicity of a certain function, it is unnecessary to know if $g \sim h$ since the original function is immediately known to be noncompletely monotonic. However, if we want to realize a given function by a combination of completely monotonic functions when the original function itself is not completely monotonic, it is necessary to know the positions of pairs which are not mutually monotonic in the expansion diagram. In this case, therefore, we have to proceed further.

C. Completely Specified Unate Functions

Sometimes in actual examples, especially when only the complete monotonicity of a given function is under consideration, mutual monotonicity of two unate functions is examined. In this case, Theorem 2.8 is simplified further.

Theorem 2.9; A necessary and sufficient condition for completely specified unate functions g and h to be mutually monotonic is to satisfy the following two conditions when they are represented by sum of prime implicants.

(a) There is no variable which appears in g and h, respectively, with opposite signs.

(b) For every pair of prime implicants a_1a_2c and a_3a_4c selected from g and h, respectively, there is either a prime implicant p of g which satisfies $a_2a_3c \rightarrow p$ or a prime implicant q of h which satisfies $a_1a_4c \rightarrow q$, where there are no common literals in a_1a_2 and a_3a_4, and all partitions a_1a_2 and a_3a_4 of selected prime implicants that satisfy $N(a_1)>0$, $N(a_3)>0$ should be considered.

Proof: In Theorem 2.8, for example, $U(\bar{a}_1a_2a_3c) \cap V(g) = \phi$ is equivalent to $U(\bar{a}_1a_2a_3c) \subseteq U(g)$. When g is unate, this also means that there is a prime implicant p of g which satisfies $\bar{a}_1a_2a_3c \rightarrow p$[88][98]. Furthermore, from the unateness of g and the fact that a_1 appears in g, $\bar{a}_1a_2a_3c \rightarrow p$ is equivalent to $a_2a_3c \rightarrow p$. We can prove this for the prime implicant q of h in a similar manner.

Q.E.D.

Of course, if $U(g) \supseteq U(h)$ is known to hold, we can add one more condition, $N(a_4)>0$, to condition (b) of Theorem 2.9.

The procedure for examining mutual monotonicity of Theorem 2.9 is almost the same as the one by Theorem 2.8. However, it is simplified in two respects: product terms considered in procedure (2) are a little simplified and the intersection property is substituted by the subsuming property.

Example 2.4:

$$g = x_1 \vee x_2x_3$$
$$h = x_2x_3. \qquad (2.27)$$

Obviously, $U(g) \supseteq U(h)$. Compare x_1 of g and x_2x_3 of h, then $a = x_1$, $a' = x_2x_3$. If we put $a_1 = x_1$, $a_3 = x_2$, $a_4 = x_3$, there is neither a prime implicant of g which is subsumed by $a_2a_3c = x_2$ nor a prime implicant of h which is subsumed by $a_1a_4c = x_1x_3$. Therefore, we have $g \nsim h$.

Example 2.5:

$$g = x_2x_3 \vee x_2x_4 \vee x_2x_5 \vee x_3x_4 \vee x_3x_5$$
$$h = x_2x_3x_4 \vee x_2x_3x_5. \qquad (2.28)$$

Obviously, $U(g) \supseteq U(h)$. Thus, considering $N(a)\geq 1$, $N(a')\geq 2$, it is necessary to examine four pairs, $\{x_2x_4, x_2x_3x_5\}$, $\{x_2x_5, x_2x_3x_4\}$, $\{x_3x_4, x_2x_3x_5\}$, and $\{x_3x_5, x_2x_3x_4\}$. Note, however, that any product term of positive literals r which satisfies $N(r)\geq 2$, except for x_4x_5, subsumes some

prime implicants of g. For the above four pairs, $c = x_2$ or $c = x_3$ and $N(a_2a_3c) \geq 2$ from $N(a_3) > 0$. Since clearly a_2a_3c is not x_4x_5, any a_2a_3c subsumes some prime implicants of g and, therefore, we obtain $g \sim h$. This technique which considers the number of literals in product terms sometimes greatly reduces the amount of computation.

The same result is also obtained by constructing a "partition diagram" for each pair of prime implicants. In the partition diagram, a_2a_3c and a_1a_4c are written in upper and lower entries, respectively, and circled if it

$x_2x_3x_5$ / x_2x_4		x_3	x_5	a_3
		x_5	x_3	a_4
x_4		x_2x_3	x_2x_5	a_2a_3c
		(circled) $x_2x_4x_5$	$x_2x_3x_4$	a_1a_4c
a_1	a_2			

$x_2x_3x_5$ / x_2x_4		x_2	x_5	a_3
		x_5	x_2	a_4
x_4		x_2x_3	x_3x_5	a_2a_3c
		(circled) $x_3x_4x_5$	$x_2x_3x_4$	a_1a_4c
a_1	a_2			

does not subsume any prime implicant of the designated function. Clearly, $g \nsim h$ if and only if there are entries such that both the upper and the lower are circled. Note that it can be shown that the partition diagrams of pairs $\{x_2x_5, x_2x_3x_4\}$ and $\{x_3x_5, x_2x_3x_4\}$ can be ommitted

since these pairs appear in the partition diagrams of $\{x_2x_4, x_2x_3x_5\}$ and $\{x_3x_4, x_2x_3x_5\}$, respectively. This property is always true. Therefore, in this example all possible partitions for these pairs which may satisfy condition (b) of Theorem 2.9 are already exhausted in the above diagrams.

2.10 Examples of the Test of Complete Monotonicity

Example 2.6: Consider an incompletely specified function and its expansion diagram shown in Fig.2.4. Although each function can be defined by f^1 and f^0, as described in Section 2.9A, it is given directly by a truth table.

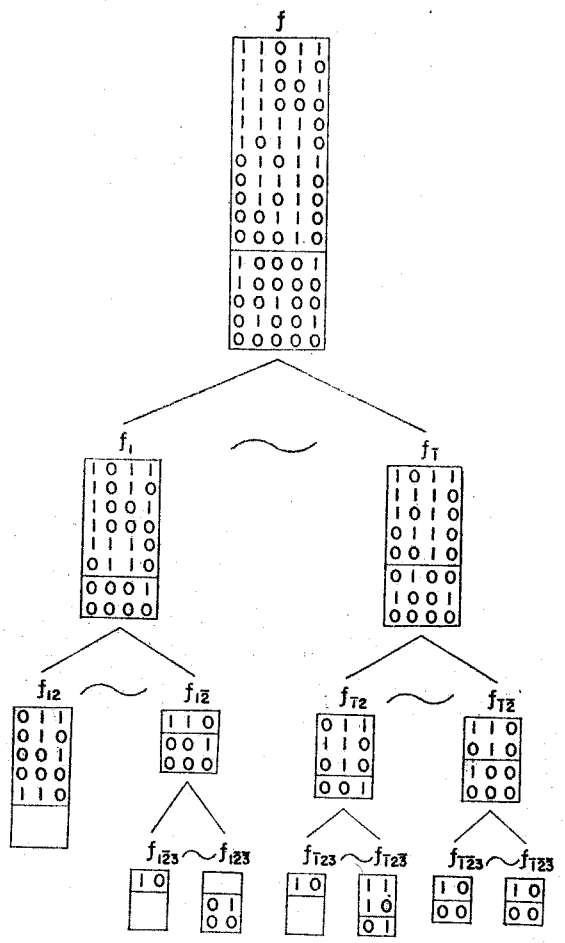


Fig.2.4 Expansion diagram of the function of Example 2.6.

The upper side of the truth table represents vectors of U(f) and the lower side represents V(f). The mutual monotonicity of f_1 and $f_{\bar{1}}$ has already been shown in Example 2.3. From Corollary 2.2A, $f_{12} \sim f_{1\bar{2}}$, $f_{1\bar{2}3} \sim f_{1\bar{2}\bar{3}}$, and $f_{\bar{1}23} \sim f_{\bar{1}2\bar{3}}$. The test of $f_{\bar{1}2} \sim f_{\bar{1}\bar{2}}$ and $f_{\bar{1}\bar{2}3} \sim f_{\bar{1}\bar{2}\bar{3}}$ can be easily performed by Definition 2.3, since the number of vectors under consideration is rather small. Consequently, all pairs of functions in the expansion diagram are mutually monotonic and, therefore, f is completely monotonic.

Example 2.7:

$$f = x_1x_2 \vee x_3x_4 \tag{2.29}$$

Expanding f, we have $f_1 = x_2 \vee x_3x_4$, $f_{\bar{1}} = x_3x_4$. These are the same type of functions which we have treated in Example 2.4 and, therefore, $f_1 \not\sim f_{\bar{1}}$. Thus f is not completely monotonic.

Example 2.8:

$$f = x_1x_2x_3 \vee x_1x_2x_4 \vee x_1x_2x_5 \vee x_1x_3x_4$$
$$x_1x_3x_5 \vee x_2x_3x_4 \vee x_2x_3x_5 \tag{2.30}$$

The expansion diagram is shown in Fig. 2.5. Since $f_{1\bar{2}} = f_{\bar{1}2}$, either function is to be expanded further. The function in the lowest level of each branch is obviously completely monotonic. By Corollary 2.2A, $f_{\bar{1}2} \sim f_{\bar{1}\bar{2}}$, $f_{\bar{1}23} \sim f_{\bar{1}2\bar{3}}$, and $f_{123} \sim f_{12\bar{3}}$. We have already

shown the mutual monotonicity of f_1 and $f_{\bar{1}}$ in Example 2.5. Since f is symmetric in $\{x_1, x_2, x_3\}$ it is not necessary to examine the mutual monotonicity of f_{12} and $f_{1\bar{2}}$ by Theorem 2.4. Thus, every pair of functions in the expansion diagram is mutually monotonic and, therefore, f is completely monotonic.

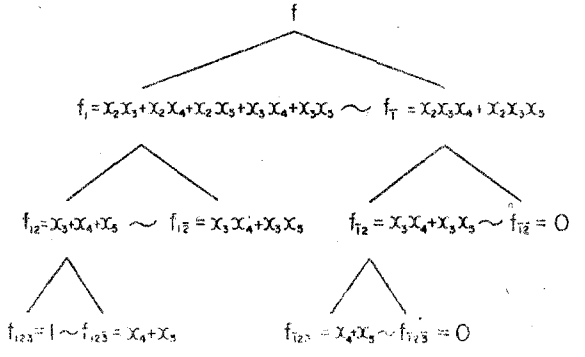


Fig. 2.5 Expansion diagram of the function of Example 2.8.

In the above expansion, the test of $f_1 \sim f_{\bar{1}}$ was somewhat complicated. This can be simplified if we employ the concept of dual functions as well as Theorem 2.5 and Theorem 2.6. One example is shown in Fig.2.6, where $f_1d_2 = ((f_1)^d)_2$, and so forth. To examine $f_1d \sim f_{\bar{1}}$, it is sufficient to consider the pair of prime implicants which satisfy $N(a) \geq 1$ and $N(a') \geq 2$, since obviously $U(f_1d) \supseteq U(f_{\bar{1}})$. However, there is no such pair of prime implicants and, therefore, we immediately have $f_1d \sim f_{\bar{1}}$. The test of the remainder is similar to the above. Consequently, we have also proved by this expansion that f is completely monotonic.

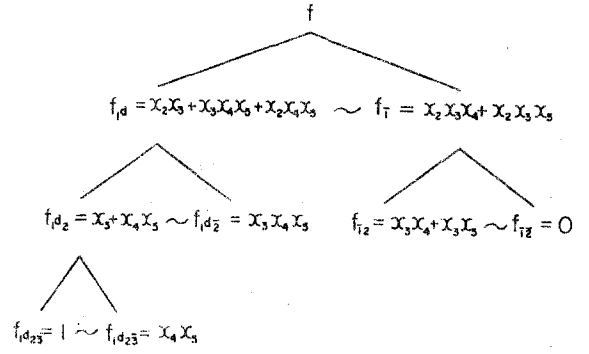


Fig.2.6 Expansion diagram of the function of Example 2.8 obtained by the aid of dualization.

As an example of a larger function, the wellknown Moor's function[98] is tested its complete monotonicity, in the reference [108]. Other useful techniques will also be found in the process for that function.

2.11 Functional Form of Completely Monotonic Functions

From Theorem 2.8 we can obtain a direct necessary and sufficient condition for complete monotonicity when the function is completely specified and represented by a sum of prime implicants. Functional forms of threshold functions have been frequently discussed,[72][74][22] but they are only necessary conditions or only sufficient conditions, not both. Therefore, the next theorem is interesting since it provides a necessary and fufficient condition, though it is of complete monotonicity.

Theorem 2.10: A necessary and sufficient condition for a completely specified function f to be completely monotonic, is to satisfy the following two conditions when it is represented by sum of prime implicants.

(a) f is unate.

(b) For every pair of prime implicants of f, $ac = a_1a_2c$ and $a'c = a_3a_4c$, such that $N(a_1)$, $N(a_2)$, $N(a_3)$, and $N(a_4)>0$, there is a prime implicant p of f which satisfies either $a_2a_3c \rightarrow p$ or $a_1a_4c \rightarrow p$, where a_1a_2 and a_3a_4 have no common variable and all possible partitions a_1a_2 and a_3a_4 should be considered.

Proof: Apply Theorem 2.8 to Theorem 2.3 condition (b). $N(a_3)>0$ and $N(a_4)>0$ are immediate consequences of $U(f) \supseteq U(f)$. Q.E.D.

Although the test of complete monotonicity by Theorem 2.10 seems more time-consuming than the previous one, it would be easier to program.

Example 2.9: Consider the function f treated in Example 2.8. As f consists of seven prime implicants, ${}_7C_2 = 21$ pairs are to be compared. However, among them, only six pairs satisfy $N(a_1) + N(a_2) \geq 2$ and $N(a_3) + N(a_4) \geq 2$ and, therefore, can satisfy $N(a_1)$, $N(a_2)$, $N(a_3)$, and $N(a_4) > 0$. They are $\{x_1x_2x_4,\ x_1x_3x_5\}$, $\{x_1x_2x_4,\ x_2x_3x_5\}$, $\{x_1x_3x_4,\ x_1x_2x_5\}$, $\{x_1x_3x_4,\ x_2x_3x_5\}$ $\{x_1x_2x_5,\ x_2x_3x_4\}$, and $\{x_1x_3x_5,\ x_2x_3x_4\}$. For example, for the first pair we have $a = x_2x_4$, $a' = x_3x_5$, and $c = x_1$. For this pair, only $\{a_2a_3c = x_1x_2x_3,\ a_1a_4c = x_1x_4x_5\}$ and $\{x_1x_2x_5,\ x_1x_3x_4\}$ should be examined for condition (b) of Theorem 2.10, considering $N(a_1)$, $N(a_2)$, $N(a_3)$, and $N(a_4) > 0$. In this case there are prime implicants $x_1x_2x_3$ and $x_1x_2x_5$ in f and one of them is subsumed by the former term of each pair. By a similar process we can eventually prove the complete monotonicity of f by this direct method.

Corollary 2.10A: When a completely specified unate function f is represented by sum of prime implicants, if for every pair of prime implicants the total number of literals which appear in either prime implicant but not both is at most 3, then f is completely monotonic.

Example 2.10:

$$f = x_1x_2x_3 \vee x_1x_2x_4x_5 \vee x_2x_3x_4. \qquad (2.31)$$

This is completely monotonic from Corollary 2.10A.

Wellknown properties which hold for functions with a small number of variables[22][76] are also easily obtained

from Theorem 2.10.

Corollary 2.10B: Unate functions of at most three variables are completely monotonic (threshold) functions.

Corollary 2.10C: Positive functions of four variables which are not completely monotonic (threshold) functions are limited to the following functions and those obtainble by permutating their variables and duals.

(1) $f = x_1x_2 \vee x_3x_4$,

(2) $f = x_1x_2 \vee x_2x_3 \vee x_3x_4$.

These corollaries can be incorporated when we are completing an expansion diagram.

2.12 Extension to Threshold Functions

Threshold functions are of practical importance since they are easily realized by physical or threshold elements. However, completely monotonic functions do not have such a practical importance and are considered to be important only as approximations of threshold functions. In this section, therefore, we consider an extension to threshold logic of concepts developed in the previous sections.

First, we introduce the concept of isobaricity corresponding to mutual monotonicity.

Definition 2.5: Consider two functions g and h and let

$$x_i \in U(g)$$
$$y_i \in V(g)$$
$$u_i \in U(h)$$
$$v_i \in V(h),$$

then g and h are "isobaric" if and only if for every set of vectors and k_1, k_2 (>0) which satisfy

$$\sum_{k_1} x_i \neq \sum_{k_1} y_i \quad \text{and} \quad \sum_{k_2} u_i \neq \sum_{k_2} v_i,$$

the following holds:

$$\sum_{k_1} x_i + \sum_{k_2} u_i \neq \sum_{k_1} y_i + \sum_{k_2} v_i,$$

where vectors are not necessarily distinct. If g and h are isobaric, they are denoted as $g \approx h$, and if not, as $g \not\approx h$.

When k_1, and k_2 are limited to 1 this concept is equivalent to mutual monotonicity, and obviously mutual monotonicity is a necessary condition for g and h to be isobaric.

The term isobaric has sometimes been used to imply that two threshold functions can be realized by the same weight vector[23]. However, in the above definition g and h can be isobaric even if g or h itself is not a threshold function. To imply the above concept we say that threshold functions g and h are "simultaneously realizable". These two concepts are equivalent when g and h are restricted to threshold functions.

Theorem 2.11: When g and h are threshold functions, g and h are simultaneously realizable if and only if g and h are isobaric.

Proof: It is known (see Muroga et al.,[72] etc.) that f is a threshold function if and only if g and h are threshold functions and simultaneously realizable, where

$$f = gx_i \vee hx_i$$

and g and h are independent of x_i. f is a threshold function if and only if f is k-asummable for any k. Then, assume g and h are not isobaric, i.e., there are vectors which satisfy

$$\sum_{k_1} x_j + \sum_{k_2} u_j = \sum_{k_1} y_j + \sum_{k_2} v_j. \qquad (2.32)$$

Adding the component x_i to the left of these vectors, (2.32) is rewritten as

$$\sum_{k_1}(1, x_j) + \sum_{k_2}(0, u_j) = \sum_{k_1}(1, y_j) + \sum_{k_2}(0, v_j),$$

and this implies f is (k_1+k_2)-summable. Thus, f is not a threshold function. The converse can also be proved in a similar manner. Q.E.D.

These concepts are meaningful since the next theorem holds and corresponds to Theorem 2.3 which is a basic property of completely monotonic functions.

Theorem 2.12: The following four conditions are equivalent.

(a) f is a threshold function.

(b) $f \approx f$.

(c) For any i ($1 \le i \le n$), $f_i \approx f_{\bar{i}}$.

(d) For a given i, both f_i and $f_{\bar{i}}$ are threshold functions and moreover, $f_i \approx f_{\bar{i}}$.

Proof: (a) $\Leftrightarrow$ (b) may be trivial. (a) $\Leftrightarrow$ (d) is a wellknown property of threshold functions, since in this case $f_i \approx f_{\bar{i}}$ implies that they are simultaneously realizable by Theorem 2.11. Therefore, we prove only (a) $\Leftrightarrow$ (c). However, (a) $\Rightarrow$ (c) is also easy to prove. (c) $\not\Rightarrow$ (a) is proved as follows. Assume that f is not a threshold function, and there are vectors such that

$$\begin{aligned} u_j &\in U(f) \\ v_j &\in V(f) \\ \sum_k u_j &= \sum_k v_j. \end{aligned} \qquad (2.33)$$

Let this be a minimal set of vectors that satisfy (2.33), i.e., no proper subset of vectors $\{u_j\} \cup \{v_j\}$ satisfies (2.33). Since all u_j can bot be the same, for some component, say x_i, the following sets are all nonempty:

$$P = \{u_j \mid (u_j)_i = 1\}$$
$$Q = \{u_j \mid (u_j)_i = 0\}$$
$$R = \{v_j \mid (v_j)_i = 1\}$$
$$S = \{v_j \mid (v_j)_i = 0\}.$$

Let #P = #R = k_1 and #Q = #S = k_2, and p_j be a vector obtained by removing the i-th component of u_j in P, etc. Then, we have

$$p_j \in U(f_i)$$
$$r_j \in V(f_i)$$
$$q_j \in U(f_i)$$
$$s_j \in V(f_i),$$
$$\sum_{k_1} p_j + \sum_{k_2} q_j = \sum_{k_1} r_j + \sum_{k_2} s_j,$$

and from the assumption of the minimal set, obviously

$$\sum_{k_1} p_j \neq \sum_{k_1} r_j \quad \text{and} \quad \sum_{k_2} q_j \neq \sum_{k_2} s_j.$$

This implies $f_i \not\equiv f_{\bar{i}}$, and thus a contradiction.

Q.E.D.

As in the previous sections, this theorem enables us to develop a similar theory of threshold functions. Note, however, that we have not yet found a simple method for testing if two given functions are isobaric. The

usefulness of isobaricity completely depends on the development of such methods.

Although complete monotonicity and realizability by a threshold element are very closely related, indeed equivalent for completely specified functions of at most 8 variables, there are certainly functions which are completely monotonic but not 1-realizable, such as Moore's function.In this case, some pairs of functions in the expansion diagram are mutually monotonic but not isobaric.

From these arguments the next theorem may be obvious.

Theorem 2.13: Completely specified completely monotonic threshold functions of at most 7 variables are isobaric (simultaneously realizable) if and only if they are mutually monotonic.

This theorem is of practical importance since the test of mutual monotonicity is rather simple, as described in the previous sections.

2.13 Classification of Threshold Functions

The number of threshold functions increases rapidly in accordance with n, though threshold functions occupy a progressively small portion of all functions. A handy method of handling those large number of functions is to classify them into fewer number of classes. These classes will be dealt with rather than individual functions in the design process, provided that a method of obtaining the original functions from such classes is known.

A convenient classification of threshold functions is provided by means of characterictic vectors.

Definition 2.6: Let

$$q_\varepsilon = \#\{ X \mid f(X) = \varepsilon \}$$
$$q^j_{\delta\varepsilon} = \#\{ X \mid x_j = \delta,\ f(X) = \varepsilon \},$$

where δ and ε assume either 0 or 1. Then the "characteristic vector" of $f(X)$ is defined by

$$C = (c_0, c_1, c_2, \ldots, c_n),$$

where

$$c_0 = q_1 - q_0$$
$$c_j = q^j_{11} + q^j_{00} - q^j_{01} - q^j_{10},$$
$$j = 1, 2, \ldots, n.$$

In other words, let the binary values be -1 and 1 instead of 0 and 1. In this case, the vector is represented by y, Then, an equivalent expression of c is

$$c = \sum_{i=1}^{2^n} f(y_i)\ (\ 1,\ y_i\ ).$$

If, conversely, we can obtain f(y) from its characteristic vector, this n+1 dimensional vector could be used as a representation of f(y). For this purpose, the one to one correspondence between function and its characteristic vector is necessary.

Theorem 2.14: A necessary and sufficient condition that a function f and its characteristic vector correspond one to one is that there is no set of vectors such that

$$u_i \in \{\ y \mid f(y) = 1\ \}$$

$$v_i \in \{\ y \mid f(y) = -1\}$$

$$\sum_{i=1}^{k} u_i = \sum_{i=1}^{k} v_i, \qquad (2.34)$$

where no repetition of vectors in the sum is permitted.

Proof: Let f and g have the same characteristec vector, i.e.,

$$\Sigma\ f(y_i)\ (\ 1,\ y_i\ ) = \Sigma\ g(y_i)\ (\ 1,\ y_i) \qquad (2.35)$$

For vectors, for which $f(y_i) = g(y_i)$ holds, the both sides of (2.35) cancels each other. Thus by changing the subscripts appropriately for the vectors with

$$f(y_i) = -g(y_i),$$

the next equation is derived from (2.35).

$$\sum_{i=1}^{k} f(y_i)\ (\ 1,\ y_i\ ) = \sum_{i=1}^{k} -f(y_i)\ (\ 1,\ y_i), \tag{2.36}$$

namely,

$$\sum_{i=1}^{k} f(y_i)\ (\ 1,\ y_i) = 0. \tag{2.37}$$

Obviously, (2.37) is equivalent to the existence of vectors given as (2.34).

Conversely, if (2.37) is satisfied for a set of vectors, we can create a function $g(y)$ by setting

$$\begin{aligned} g(y_i) &= -f(y_i),\ i = 1,2,\ldots,k \\ g(y_i) &= f(y_i),\ \text{otherwise.} \end{aligned} \tag{2.38}$$

This $g(y)$ satisfies (2.35) and therefore has the same characteristec vector as $f(y)$. Q.E.D.

Let us compare this result with the asummability property (2.6) of threshold functions, and also with the 2-asummability property of completely monotonic functions (Definition 2.1).

Let us denote the class of threshold functions by $\mathcal{T}$, the class of completely monotonic functions by $\mathcal{M}$, and the class of functions which correspond one to one with their characteristic vectors by $\mathcal{C}$, then the next theorem follows immediately.

Theorem 2.15:

$$\mathcal{T} \subseteq \mathcal{C} \subseteq \mathcal{M}$$

The relation $\mathcal{T} \subseteq \mathcal{M}$ was first proved by Chow[14]. It is also known that $\mathcal{T} = \mathcal{M}$ holds for functions of up to 8 variables[100][79]. Those functions which belong to $\mathcal{M}$ but not $\mathcal{T}$ have been constructed by various people[98][27][114]. Among those are a function of 9 variables which belongs to $\mathcal{C}$ but not $\mathcal{T}$, and a function of 15 variables which belongs to $\mathcal{M}$ but not $\mathcal{C}$. Both functions were constructed by Gabelman[27] for a different purpose.

As a result, we are able to use the characteristic vector instead of the function if it is known to be a threshold function, because Theorem 2.15 asserts that $\mathcal{T} \subseteq \mathcal{C}$, i.e., the characteristic vector determines a function uniquely. Several methods are available[20][56][104], to obtain the original function from its characteristic vector, if a given function is a threshold function. When completely monotonic functions are concerned, however, the one to one correspondence between functions and their characteristic vectors does not hold in general, except for functions of up to

8 variables.

Now, let us denote the characteristic vector of $f(y)$ by

$$\mathrm{ch}.f(y) = (c_0, c_1, \ldots, c_n) \qquad (2.39)$$

Then the next properties immediately follow from its definition.

(1) Let σ be a permutation. Then

$$\mathrm{ch}.f(y_{\sigma(1)}, y_{\sigma(2)}, \cdots, y_{\sigma(n)}) = (c_0, c_{\sigma(1)}, \cdots, c_{\sigma(n)}). \qquad (2.40)$$

(2)

$$\mathrm{ch}.f(y_1, \ldots, y_{i-1}, \bar{y}_i, y_{i+1}, \ldots, y_n) = (c_0, \ldots, c_{i-1}, -c_i, c_{i+1}, \ldots, c_n), \qquad (2.41)$$

where $\bar{y}_i$ is the negation of y_i, i.e.,

$$\bar{y}_i = -y_i.$$

(3)

$$\mathrm{ch}.\bar{f}(y) = (-c_0, -c_1, \ldots, -c_n). \qquad (2.42)$$

(4)

$$\mathrm{ch}.f^{d}(y) = (-c_0, c_1, c_2, \ldots, c_n). \qquad (2.43)$$

(5)

$$\mathrm{ch}.y_j f(y_1 y_j, \ldots, y_{j-1} y_j, y_j, y_{j+1} y_j, \ldots, y_n y_j) = (c_j, c_1, \ldots, c_{j-1}, c_0, c_{j+1}, \ldots, c_n). \qquad (2.44)$$

Property (5) is first proved by Dertouzos[20], in conjunction with the classification of threshold functions.

These properties can be used in classifying functions of class $\mathcal{C}$ by means of their characteristic vectors, because the next theorem holds.

Theorem 2.16: $\mathcal{C}$ is closed under the 5 operations shown in (2.40) - (2.44).

Proof: A proof is given for the negation of variables, i.e., (2.41) in the above list of properties. Other cases will be proved similarly.

Let g be f with y_j negated:

$$g(y) = f(y_1, \ldots, \bar{y}_j, \ldots, y_n).$$

If f(y) does not belong to $\mathcal{C}$, there exist a set of vectors

$$u_i \in \{ y \mid f(y) = 1 \}$$

$$v_i \in \{ y \mid f(y) = -1 \}$$

and

$$\sum_{i=1}^{k} u_i = \sum_{i=1}^{k} v_i ,$$

from Theorem 2.14. Now let us define vectors u_i' and v_i' as follows;

$$u_i' = (u_1, \ldots, u_{j-1}, \bar{u}_j, u_{j+1}, \ldots, u_n)$$

$$v_i' = (v_1, \ldots, v_{j-1}, \bar{v}_j, v_{j+1}, \ldots, v_n) .$$

Obviously,

$$g(u_i') = 1$$

$$g(v_i') = -1,$$

and

$$\sum_{i=1}^{k} u_i' = \sum_{i=1}^{k} v_i'.$$

This means that $g(y)$ does not belong to $\mathcal{C}$ either. The converse can be similarly proved. Q.E.D.

As a result of this theorem, it is now possible to classify those functions in $\mathcal{C}$, by defining a canonical form of characteristic vector. For example, let us take the absolute values of components of C (properties (2)(3)(4)) and then arrange them in the descending order (properties (1)(5)). The resultant vector is defind as the canonical form of characteristic vector. Now, it is defined that functions with the same canonical form of characteristic vector belong to the same class of functions.

For functions in the class $\mathcal{T}$, this calssification was actually applied and the table of all the canonical characteristic vectors is available[100] for threshold functions of up to 7 variables. (For these functions, in fact, $\mathcal{T} = \mathcal{C}$ holds as mentioned above.)

In this section, only completely specified functions were considered. A more general discussion on incompletely specified functions and the case in which a probability is associated with each input vector is given by Yajima and Ibaraki[105].

2.14 Conclusion

The concept of mutual monotonicity was introduced and investigated in this chapter. Mutual monotonicity is of importance because it gives a necessary and sufficient condition for complete monotonicity and, moreover, it can be tested simply by algebraic manipulation. Complete monotonicity is in some cases a necessary and sufficient condition for 1-realizability, and therefore, these concepts can be applied to the test of 1-realizability.

Since most functions are not completely monotonic (1-realizable)[10][98][103][92][78][112], there is an acute need for developing a theory of compound synthesis by completely monotonic functions or, if possible, by threshold functions. The concept of mutual monotonicity and the expansion diagram yields a method of economical compound synthesis, though it does not necessarily produce a minimal network. These topics will be dealt with in the next chapter.

It was shown that if we employ isobaricity instead of mutual monotonicity, the whole theory could be extended to threshold functions in parallel, and the same holds for the above-mentioned method of compound synthesis. However, simple testing methods of isobaricity have not been found yet, and the usefulness of the concept of isobaricity depends on their development.

Considering that completely monotonic functions are 2-asummable functions and threshold functions are k-asummable functions for any k, it is possible to

develop a similar theory of subsets of all functions, i.e., k-asummable functions for some k. However, this concept does not exhibit such practical importance at present as 2-asummability, which can be checked easily by algebraic manipulation, and as k-asummability for any k, which is equivalent to 1-realizability.

The idea of mutual monotonicity and isobaricity was extended to the case of multi-threshold threshold functions in [116].

Chapter 3. Network synthesis Using Completely Monotonic (Threshold) Functions

3.1 Introduction

As briefly reviewed in Section 1.3, there has been no satisfactory synthesis method which is easy to carry out and yet can yield reasonably economical networks. In this chapter, we will use the concept of mutual monotonicity and expansion diagram, developed in Chapter 2, for the network synthesis. By examining non-mutually monotonic pairs of functions in the expansion diagram, an algorithm for obtaining a network of completely monotonic functions which realizes the required function f is devised.

The network could be further simplified by applying procedures also considered in this chapter. The whole process can be easily performed by hand computation for functions of moderate number of variables. It always yields quite economical networks, not necessarily optimal, in the author's experience, provided that the simplification techniques are adequately applied.

Therefore, it is believed that this algorithm constitutes one of the major approaches of network design of threshold (completely monotonic) gates, when practical algorithms are required.

However, the method is concerned with the compound synthesis using completely monotonic functions, and strictly speaking, there are completely monotonic functions which are not threshold functions. Thus it is

necessary to extend the method to threshold logic. In Section 3.8, it will be shown that this is made possible by replacing the complete monotonicity by the 1-realizability and the mutual monotonicity by the isobaricity.

3.2 Expansion Diagram

It is necessary to obtain an expansion diagram of a given function in order to realize a function by a combination of completely monotonic functions. In this case, however, the position of each non-mutually monotonic pair is essential and all such pairs must be found. The total number of non-mutually monotonic pairs changes according to the sequence of expansion variables and it is desirable to find an expansion diagram with the minimum number of such pairs.

Example 3.1: Consider a completely specified function:

$$f = x_1\bar{x}_2x_3x_6 \vee x_1\bar{x}_2x_4x_6 \vee x_1x_3x_4x_6$$
$$\vee x_1\bar{x}_2x_5x_6 \vee x_1x_3x_5x_6 \vee \bar{x}_2x_3x_4x_5x_6$$
$$\vee x_2\bar{x}_3\bar{x}_6 \vee x_2x_4\bar{x}_6 \vee \bar{x}_1x_2x_5\bar{x}_6. \qquad (3.1)$$

First, if we successively expand this function by variables x_1, x_2, x_3,... , the expansion diagram of Fig.3.1(a) is obtained. Next, if it is expanded in the sequence x_6, x_1, x_2,... , the expansion diagram is the one shown in Fig.3.1(b). The total number of non-mutually monotonic pairs is reduced from 4 to 1.

Several notations concerning an expansion diagram must be defined here. If a function g is just below a function f and connected by a branch, g is said to be a "first-generation successor" of f. Namely, f_{ai} is a first-generation successor of f_a. If h is a first-generation

successor of g and g is a k-th-generation successor of f, then h is a (k+1)st-generation successor of f. A successor of some generation is simply called a "successor". Obviously, every function in an expansion diagram except the bottom ones has two first-generation successors. If two first-generation successors g and h of f are non-mutually monotonic, this non-mutually monotonic pair is said to be induced by f.

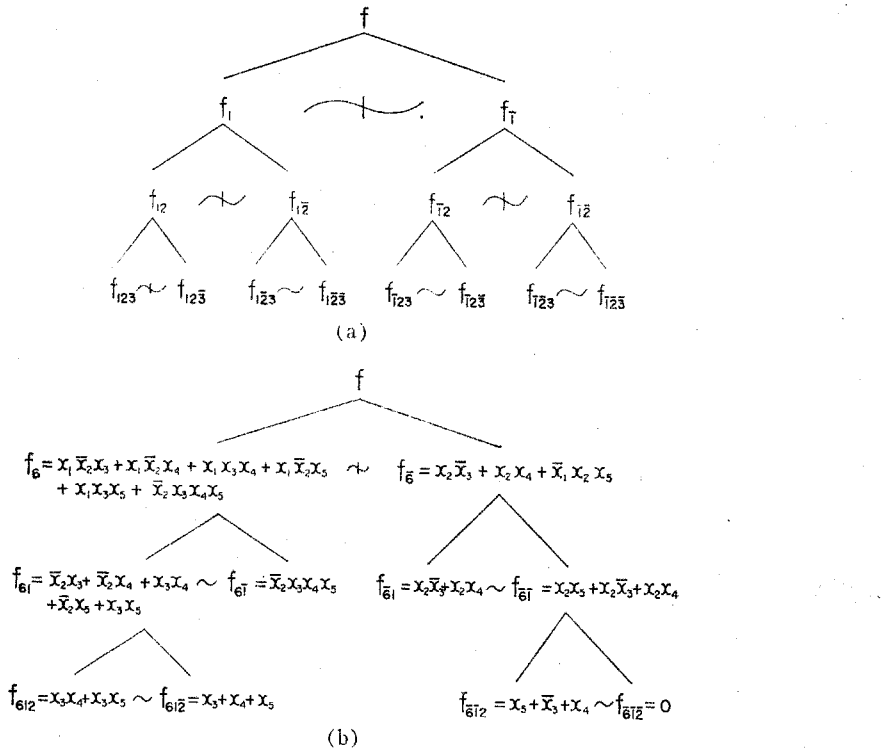


Fig.3.1. Two expansion diagrams of the function in Example 3.1.

3.3 Fundamental Theorem for the Realization of an Arbitrary Function

If an expansion diagram of a certain function contains no non-mutually monotonic pair, it is completely monotonic and can be trivially realized by one completely monotonic function. In most cases, however, an expansion diagram contains non-mutually monotonic pairs and these pairs must be removed in order to realize it by completely monotonic functions. In this chapter, this is done by assigning an augmented variable to every non-mutually monotonic pair and thus equivalently making it mutually monotonic. The next problem is how to realize the augmented variables by a combination of completely monotonic functions.

Before stating the next theorem which is basic to these problems, we define two "constant functions" 1_i and 0_i associated with f_i and negation of f_i as follows:

$$\begin{aligned} &1_i\colon U(1_i) = U(f_i) \cup V(f_i), && V(1_i) = \phi \\ &0_i\colon U(0_i) = \phi, && V(0_i) = U(f_i) \cup V(f_i) \\ &\bar{f}_i\colon U(\bar{f}_i) = V(f_i), && V(\bar{f}_i) = U(f_i). \end{aligned} \tag{3.2}$$

Theorem 3.1: Let a function f of n variables be expanded as

$$f = f_i x_i \vee f_{\bar{i}} \bar{x}_i \tag{3.3}$$

and suppose that f_i and $f_{\bar{i}}$ are both completely monotonic but $f_i \not\propto f_{\bar{i}}$. For each of the eight augmented variables below,

$$\alpha = \begin{cases} f_i \\ \bar{f}_i \end{cases} x_i + \begin{cases} 1_i \\ 0_i \end{cases} \bar{x}_i \qquad (3.4)$$

or

$$= \begin{cases} 1_i \\ 0_i \end{cases} x_i + \begin{cases} f_i \\ \bar{f}_i \end{cases} \bar{x}_i, \qquad (3.5)$$

consider the function of (n+1) variables $f'(x_1,\ldots,x_n, \alpha(x_1,\ldots, x_n))$ which satisfies $f' = f$ when f' is considered as a function of X. Then f' is completely monotonic with respect to $x_1, x_2,\ldots, x_n, \alpha$, and, moreover, α is also completely monotonic with respect to $x_1,\ldots, x_n$.

Proof: We prove only for an augmented variable

$$\alpha = f_i x_i \vee 0_i \bar{x}_i.$$

Others can be similarly proved. From the assumption of Theorem 3.1, f_i is completely monotonic and 0_i is a constant function. Then, $f_i \vee 0_i$ is directly obtained from Corollary 2.2A and it implies that α is completely monotonic from Theorem 2.3, condition (d).

To prove the compelete monotonicity of f', assume f' is not completely monotonic. Then we have four vectors such that

$$\begin{aligned} &u_1', u_2' \in U(f') \\ &v_1', v_2' \in V(f') \\ &u_1' + u_2' = v_1' + v_2' \end{aligned} \qquad (3.6)$$

from the definition. Here note that $\alpha = 1$ if and only if

$f_i = x_i = 1$, i.e., $\alpha(x) = 1$ only for $x = (a, \overset{x_i}{1})$, where $f_i(a) = 1$. Namely, the α components of v_1' and v_2' are both 0 because the vector $(a, \overset{x_i}{1})$, where $f_i(a) = 1$, belongs to $U(f)$ and accordingly $(a, \overset{x_i}{1}, \overset{\alpha}{1})$ belongs to $U(f')$. This implies that the α components of u_1' and u_2' are also 0, since (3.4) holds. Thus the x_i components of u_1' and u_2' are 0 because if $\alpha = 0$ and $x_i = 1$, u_1' and u_2' must belong to $V(f')$. Equation (3.6) again implies the x_i components of v_1' and v_2' are 0. Thus letting the u_1', u_2', v_1' and v_2' from which x_i and α components are eliminated be u_1, u_2, v_1, and v_2, (3.6) is rewritten as

$$\begin{aligned} &u_1, u_2 \in U(f_i) \\ &v_1, v_2 \in V(f_i) \\ &u_1 + u_2 = v_1 + v_2. \end{aligned} \tag{3.7}$$

This is equivalent to $f_{\tilde{i}}$ not being completely monotonic, contradicting the assumption. Q.E.D.

Thus, by Theorem 3.1 we can remove the non-mutually monotonic pair $f_i \not\propto f_{\tilde{i}}$ equivalently if we consider a variable α together. Note that the function f' is no longer completely specified even if f is completely specified.

The eight kinds of augmented variables for a non-mutually monotonic pair $g \not\propto h$ are illustrated in Fig.3.2. In the figure, the output value of α for each of four sets of input vectors U(g), V(g), U(h), and V(h) are

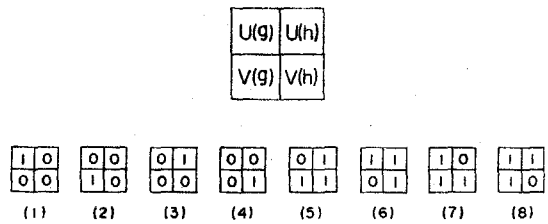


Fig.3.2. Eight augmented variables permitted by Theorem 3.1.

written in the manner designated above. For example, when g and h are written as f_i and $f_{\bar{i}}$, respectively, Fig. 3.2 (1) shows $\alpha = f_i x_i \vee 0_{\bar{i}} \bar{x}_i$. Obviously, the α of Theorem 3.1 is to assign an output value 1 to one or three of these four sets of input vectors. This α is said to be given to the non-mutually monotonic pair $f_i \not\propto f_{\bar{i}}$ or f which induces $f_i \not\propto f_{\bar{i}}$.

These augmented variables may be divided into two classes, one expressed by (3.4) and the other by (3.5). In the former case f_i is said to be "essential" to α and in the latter case $f_{\bar{i}}$ is essential.

Example 3.2: Expand an incompletely specified function f of three variables as shown in Fig.3.3(a), and then Fig.3.3 (b) is obtained. Obviously, though f_1 and $f_{\bar{1}}$ are both completely monotonic, $f_1 \not\propto f_{\bar{1}}$ holds and f is not completely monotonic, If we attach an augmented variable

$$\alpha = f_1 x_1 \vee 0_{\bar{1}} \bar{x}_1, \qquad (3.8)$$

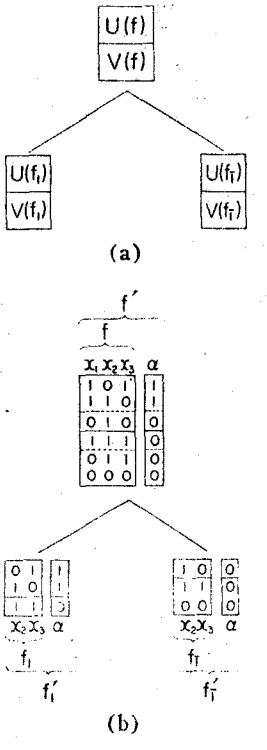


Fig.3.3. Expansion diagram and augmented variable of the function in Example 3.2.

the resulting f' is a function of four variables as shown in level 3 of Fig. 3.3 (b). Of course f' is completely monotonic.

Example 3.3: Condider the function f of Example 3.1. If we adopt the latter expansion (Fig.3.1 (b)), there is only one non-mutually monotonic pair f_6 & $f_{\bar{6}}$, and

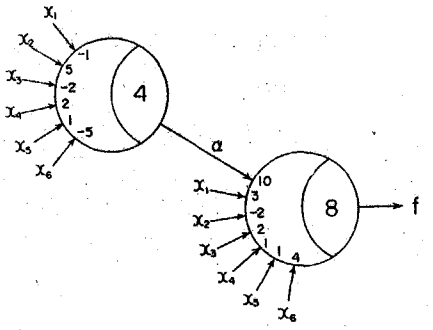


Fig.3.4. Realization of the function in Example 3.3 by threshold elements.

both f_6 and $f_{\bar{6}}$ are completely monotonic. Thus, Theorem 3.1 can be applied directly. The augmented variable α may be any one of eight possibilities. For example, let $\alpha = 0_6 x_6 \vee f_{\bar{6}} \bar{x}_6$, then f is realized by a combination of two completely monotonic (in this case, threshold) functions, one to realize α and the other to realize f' (= f) itself. This is illustrated in Fig. 3.4.

3.4 General Case

In general, there is more than one non-mutually monotonic pair in the expansion diagram of a given function and therefore Theorem 3.1 cannot be applied directly. However, functions of the lower level of each branch in the expansion diagram become necessarily completely monotonic and the assumption of Theorem 3.1 is satisfied in this part. Therefore, if we remove non-mutually monotonic pairs successively from the lower level by attaching augmented variables, we have eventually one completely monotonic function of more than n variables which equivalently realizes the given original function.

Before stating a rigorous procedure, assume that a function g is expanded by x_i and $g_i \propto g_{\bar{i}}$ holds, (i.e., g induces the non-mutually monotonic pair $g_i \propto g_{\bar{i}}$) and let α be an augmented variable given to this non-mutually monotonic pair. Then it follows from the proof of Theorem 3.1 that if g_i is essential to α then α is completely monotonic if and only if g_i is completely monotonic, and $g'(= g)$ is completely monotonic if and only if $g_{\bar{i}}$ is completely monotonic. Therefore, the augmented variable α and the function g' are realized as shown in Fig.3.5(a), provided that augmented variables $\alpha_1, \alpha_2, \ldots, \alpha_{k_i}$ and $\beta_1, \beta_2, \ldots, \beta_{k_{\bar{i}}}$ are necessary for g_i and $g_{\bar{i}}$, respectively, to become completely monotonic, and g_i is essential to α. If $g_{\bar{i}}$ is essential to α, the realization is shown in Fig.3.5(b). For the former realization, $\alpha, \beta_1, \ldots, \beta_{k_{\bar{i}}}$ are

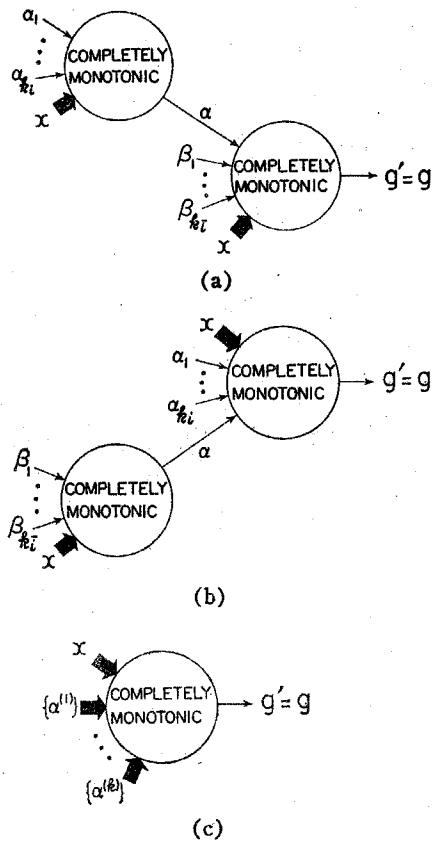


Fig.3.5. Realization of a function which has non-mutually monotonic pairs.

said to be "necessary" for g and for the latter realization, α, $\alpha_1, \ldots, \alpha_{k_i}$ are necessary for g.

When g does not induce a non-mutually monotonic pair, suppose that $g^{(1)}, \ldots, g^{(k)}$ are all the nearest successors of g which induce non-mutually monotonic pairs (i.e., all paths from g downward either terminate at

one of $g^{(1)}$ through $g^{(k)}$ or have no function which induces a non-mutually monotonic pair) and sets of augmented variables $\{\alpha^{(1)}\},\ldots,\{\alpha^{(k)}\}$ are necessary for $g^{(1)},\ldots,g^{(k)}$, respectively. Then the function

$$g'(x,\{\alpha^{(1)}\},\cdots,\{\alpha^{(k)}\})$$

which satisfies g' = g is completely monotonic. This is true because if $\{\alpha^{(1)}\},\ldots,\{\alpha^{(k)}\}$ are taken into consideration, $g^{(1)},\ldots,g^{(k)}$ can be considered to be completely monotonic and thus the expansion diagram does not contain non-mutually monotonic pairs any more. In this case, augmented variables

$$\{\alpha^{(1)}\}\cup\{\alpha^{(2)}\}\cup\cdots\cup\{\alpha^{(k)}\} \qquad (3.9)$$

re said to be necessary for g and g is realized as hown in Fig.3.5(c).

Then the procedure is to produce each augmented variables recursively from the lower level, and finally realize the given f by either one of the methods shown above. A procedure of realizing an arbitrary function f of n variables is described.

(1) Find in the expansion diagram of f a function which induces a non-mutually monotonic pair and which satisfies one of the following:
   (a) It has no successor which induces non-mutually monotonic pair,
   (b) It has successors which induce non-mutually monotonic pairs but for which all necessary augmented variables are already determined.

Let this function be g, and suppose that g is expanded by x_i and $g_i \not\propto g_{\bar{i}}$. Let $\alpha_1, \ldots, \alpha_{k_i}$ be necessary for g_i and $\beta_1, \ldots, \beta_{k_i}$ for g_i. Then realize an augmented variable α for this pair. If we are to obtain an augmented variable α to which g_i is essential, it can be realized as shown in Fig. 3.6(a). In case of g_i being essential, the realization is shown in Fig. 3.6(b). In the former case, $\alpha, \beta_1, \ldots, \beta_{k_{\bar{i}}}$ are necessary for g, and in the latter case, $\alpha, \alpha_1, \ldots, \alpha_{k_i}$ are necessary for g. Repeat this procedure until there is no function which induces a non-mutually pair except for f itself.

(2) Realize f according to the method described above (Fig.3.5(a),(b),(c)). Consider g as f in this case.

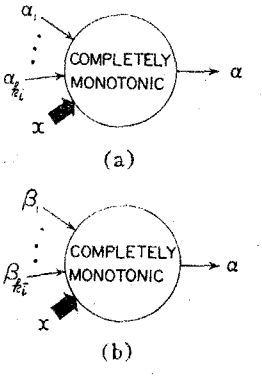


Fig.3.6. Realization of an augmented variable.

Note that in the above procedure an augmented variable for each function which induces a non-mutually monotonic pair is determined recursively but it is not necessary to realize each individual function (g in procedure (1)) except for the given function f. Therefore if there are P non-mutually monotonic pairs, we need at most P augmented variables and thus P completely monotonic functions interconnected to realize these augmented variables. One other completely monotonic function which realizes the given function f itself is also necessary.

Theorem 3.2: If there are P non-mutually monotonic pairs in the expansion diagram of a given function f of n variables, f can be realized by a combination of at most (P + 1) completely monotonic functions within n levels. The number of input variables of each completely monotonic function is not more than n + P.

Example 3.4: Consider the following completely specified function of five variables;

$$f = x_1\bar{x}_2\bar{x}_3\bar{x}_4x_5 \vee \bar{x}_1x_2\bar{x}_3\bar{x}_5 \vee x_1\bar{x}_2x_3\bar{x}_5 \vee \bar{x}_1\bar{x}_2\bar{x}_3x_4 \vee x_2x_3x_4x_5. \qquad (3.10)$$

The expansion diagram is in Fig.3.7. In this case P = 5 and by Theorem 3.2, f can be realized by at most P + 1 = 6 completely monotonic functions. First, procedure (1) is applied to three functions, f_{12}, $f_{\bar{1}2}$, and $f_{\bar{1}\bar{2}}$. Augmented variables necessary for f_{12}, $f_{\bar{1}2}$, and $f_{\bar{1}\bar{2}}$ are of the type $g_ix_i \vee 0_i\bar{x}_i$, denoted α_{12}, $\alpha_{\bar{1}2}$, and $\alpha_{\bar{1}\bar{2}}$, respectively.

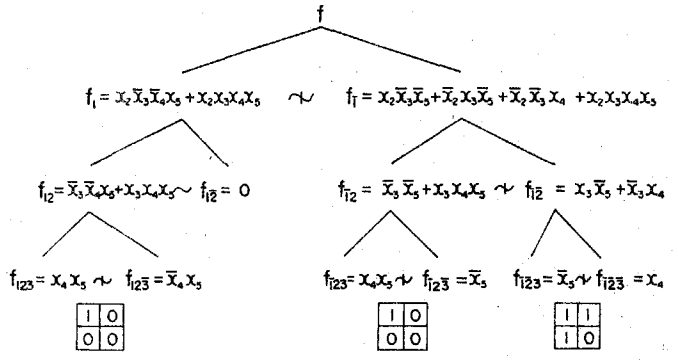


Fig.3.7. Expansion diagram of the function in Example 3.4.

Procedure (1) is then applied to $f_{\bar{1}}$ also which induces a non-mutually monotonic pair $f_{\bar{1}2} \sim f_{\bar{1}\bar{2}}$. The necessary variables for $f_{\bar{1}2}$ and $f_{\bar{1}\bar{2}}$ have already been determined. Now suppose that the augmented variable $\alpha_{\bar{1}}$ is obtained as

$$\alpha_{\bar{1}} = f_{\bar{1}2}x_2 \vee 0_{\bar{1}\bar{2}}\bar{x}_2.$$

Since $f_{\bar{1}2}$ is essential, the necessary augmented variables for $f_{\bar{1}}$ are $\alpha_{\bar{1}}$ and $\alpha_{\bar{1}\bar{2}}$. There is no additional function to which procedure (1) can be applied except f itself. The augmented variable to f, i.e., to $f_1 \sim f_{\bar{1}}$, is determined as

$$\alpha = f_1x_1 \vee 0_1\bar{x}_1, \qquad (3.11)$$

then the necessary augmented variables for f are α and the essential variables for $f_{\bar{1}}$, $\alpha_{\bar{1}}$ and $\alpha_{\bar{1}\bar{2}}$. Augmented

variable α itself needs α_{12} (see Fig.3.6). The whole network is shown in Fig.3.8, where six elements are used. If we adopt different types of augmented variables, the resulting network becomes different. However, these realizations may not be very economical. Methods of simplification will be considered in the subsequent sections.

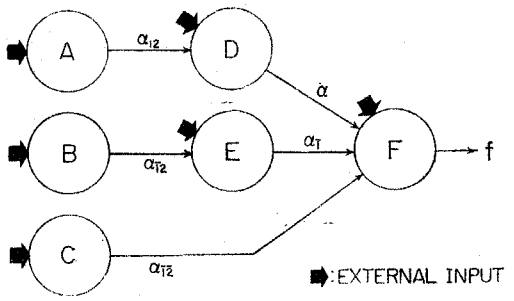


Fig.3.8. Realization of the function in Example 3.4 by completely monotonic functions.

3.5 Simplification by Uniting Augmented Variables

In this and subsequent sections, we shall discuss simplification methods of realization obtained by the direct application of the previous procedure. First, a simplification is described which can be applied easily and is usually quite effective. For example, consider two augmented variables α_{12} and $\alpha_{\bar{1}2}$ in Example 3.4. These are defined* on disjoint sets of imput vectors such that

$$\alpha_{12}: U(\alpha_{12}) \cup V(\alpha_{12}) = \{x \mid x_1 = 1, x_2 = 1\}$$
$$\alpha_{\bar{1}2}: U(\alpha_{\bar{1}2}) \cup V(\alpha_{\bar{1}2}) = \{x \mid x_1 = 0, x_2 = 1\}.$$

Thus, if a new function β obtained by uniting the above two functions α_{12} and $\alpha_{\bar{1}2}$, i.e.,

$$\beta: U(\beta) = U(\alpha_{12}) \cup U(\alpha_{\bar{1}2})$$
$$V(\beta) = V(\alpha_{12}) \cup V(\alpha_{\bar{1}2})$$

is completely monotonic, this one completely monotonic function β can provide us simultaneously with two augmented variables α_{12} and $\alpha_{\bar{1}2}$.

More generally, for functions $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(k)}$ defined on disjoint sets $X^{(1)}, X^{(2)}, \ldots, X^{(k)}$ of the same variables, let α be a function defined on the set $X^{(1)} \cup X^{(2)} \cup \ldots \cup X^{(k)}$ which satisfies

---

*Here we may also consider α_{12} as defined on a subset of the set $\{y \mid (1, 1, y) \in \{x\}\}$, thus reducing the dimension of vectors under consideration. However, hereafter we use the first definition for the convenient development of theory.

$$\alpha \mid X^{(j)} = \alpha^{(j)} \qquad j = 1, 2, \cdots, k,$$

where $\alpha|X^{(j)}$ is the function α restricted to the set $X^{(j)}$. We denote a function possessing this property as

$$\alpha = \alpha^{(1)} \cup \alpha^{(2)} \cup \cdots \cup \alpha^{(k)}. \tag{3.12}$$

Now the probems are,first, how to select candidates from non-mutually monotonic pairs in an expansion diagram which may be united, and second, how to determine the augmented variables from the eight possibilities permitted by Theorem 3.1 so as to make the united function completely monotonic,

For the first problem, comsider two functions in the diagram, one of which is a successor of the other. This means that one function is obtained from the other by fixing some variables, i.e., these two functions do not have disjoint sets of input vectors; moreover, if both functions induce non-mutually monotonic pairs, the augmented variable to one pair is required for the other pair. This leads to the following theorem.

Theorem 3.3: The set of augmented variables which may be united satisfies necessarily the condition that none of the functions to which the augmented variables are assigned is a successor of the other.

For example, any augmented variables in the same level may be united and augmented variables such as $\alpha_{\bar{1}}$ and $\alpha_{\bar{1}2}$ in Example 3.4 cannot be united, according to this theorem.

However, the complete monotonicity of a united function must ,in general, be examined by the method described in Chapter 2. If we are to obtain the most economical one it is necessary to examine all possible configurations of augmented variables which satisfy Theorems 3.1 and 3.3. Thus, it is desirable to find a simple necessary and sufficient condition for a united function to be completely monotonic. Although such a simple necessary and sufficient condition has not been found yet, a necessary condition given in Theorem 3.4 will reduce the amount of computation greatly and a sufficient condition, to be given in Theorem 3.5, is also useful when augmented variables satisfy the assumption in the theorem.

Theorem 3.4: Let $\alpha = \alpha^{(1)} \cup \alpha^{(2)} \cup \ldots \cup \alpha^{(k)}$. If α is completely monotonic, then so are $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(k)}$; moreover, for any i and j, $\alpha^{(i)} \sim \alpha^{(j)}$ holds.

Proof: From the definition of α, it is obvious that $\alpha^{(1)}, \ldots, \alpha^{(k)}$ are all completely monotonic. To prove $\alpha^{(i)} \sim \alpha^{(j)}$, assume $\alpha^{(i)} \not\sim \alpha^{(j)}$. This implies the existence of the following four vectors,

$$u_1 \in U(\alpha^{(i)}), \qquad u_2 \in U(\alpha^{(j)})$$
$$v_1 \in V(\alpha^{(i)}), \qquad v_2 \in V(\alpha^{(j)})$$
$$u_1 + u_2 = v_1 + v_2.$$

However, $u_1, u_2 \in U(\alpha)$ and $v_1, v_2 \in V(\alpha)$ obviously and this implies α is not completely monotonic. Thus a contradiction. Q.E.D.

Disjoint sets $X^{(1)}$, $X^{(2)}$,..., $X^{(k)}$ of interest in this chapter are as follows: for each $X^{(j)}$, there is a subvector $a^{(j)}$ such that

$$X^{(j)} = \{x \mid x = (a^{(j)}, y)\}.$$

Let $s^{(1)}$, $s^{(2)}$,..., $s^{(k)}$ be subvectors of

$$a^{(1)}, a^{(2)}, \cdots, a^{(k)},$$

respectively, which are restricted to the common variables of $a^{(1)}$, $a^{(2)}$,..., $a^{(k)}$. Then it is convenient to write

$$\alpha^{(j)} = s^{(j)}\gamma^{(j)}, \qquad (3.13)$$

where $s^{(j)}$ is a product term of literals which corresponds to $s^{(j)}$ and $\gamma^{(j)}$ is a function of m-N($s^{(j)}$) variables, where m is the dimension of $\alpha^{(j)}$. Theorem 3.4 is rewritten as follows.

Corollary 3.1: Let $\gamma^{(1)}$, $\gamma^{(2)}$,..., $\gamma^{(k)}$ be functions obtained from a completely monotonic function α of Theorem 3.4, then $\gamma^{(1)}$,..., $\gamma^{(k)}$ are all completely monotonic and for any i,j, $\gamma^{(i)} \sim \gamma^{(j)}$ holds.

Since the complete monotonicity of each augmented variable is guaranteed by Theorem 3.1, it is only necessary to select a set of pairwise mutually monotonic augmented variables. The converse of Theorem 3.4, however, is not always true and it is still necessary to examine the complete monotonicity of a united function.

One of the particular cases in which the converse of

Theorem 3.4 holds is discussed in the next theorem.

Theorem 3.5: Let α, $\alpha^{(j)}$, $\gamma^{(j)}$, $s^{(j)}$ $(j = 1, 2, \ldots, k)$ be defined as above. If for every distinct j_1, j_2, j_3, j_4,

$$\begin{aligned} s^{(j_1)} &\neq s^{(j_2)} \\ s^{(j_1)} + s^{(j_2)} &\neq s^{(j_3)} + s^{(j_4)} \end{aligned} \tag{3.14}$$

hold, the next three condition are equivalent.

(a) α is completely monotonic.

(b) $\alpha^{(j)}$, $j = 1, 2, \ldots, k$, are completely monotonic and for any j_1, j_2, $\alpha^{(j_1)} \sim \alpha^{(j_2)}$ holds.

(c) $\gamma^{(j)}$, $j = 1, 2, \ldots, k$, are completely monotonic and for any j_1, j_2, $\gamma^{(j_1)} \sim \gamma^{(j_2)}$ holds.

Proof: For simplicity, we prove (a) $\Leftrightarrow$ (b) only. (a) $\Rightarrow$ (b) was already proved by Theorem 3.4. In order to prove (b) $\Rightarrow$ (a), assume α is not completely monotonic. Then there are four vectors such that

$$\begin{aligned} u_1, u_2 &\in U(\alpha), \quad v_1, v_2 \in V(\alpha) \\ u_1 + u_2 &= v_1 + v_2. \end{aligned} \tag{3.15}$$

Let $u_1 = (s^{(j_1)}, u_1')$, $u_2 = (s^{(j_2)}, u_2')$, $v_1 = (s^{(j_3)}, v_1')$, and $v_2 = (s^{(j_4)}, v_2')$; then from assumption (3.14) $s^{(j_1)}$, $s^{(j_2)}$, $s^{(j_3)}$, $s^{(j_4)}$ are not all distinct. First suppose $s^{(j_1)} = s^{(j_2)}$, then we have $s^{(j_1)} = s^{(j_2)} = s^{(j_3)} = s^{(j_4)}$ from (3.15). This implies that $\alpha^{(j_1)}$ itself is not completely monotonic. Next suppose that $s^{(j_1)} = s^{(j_3)}$, then $s^{(j_2)} = s^{(j_4)}$ is derived from (3.15). This implies $\alpha^{(j_1)} \propto \alpha^{(j_2)}$. Finally,

either assumption leads to a contradiction.

Q.E.D.

Therefore, if S vectors of given augmented variables satisfy assumption (3.14) of Theorem 3.5, it is not necessary to examine the complete monotonicity of the united function again. For example, let Fig.3.9 be part of a certain expansion diagram and α_a is given to g_a, then S vectors of $\{\alpha_i, \alpha_{\bar{i}}\}$ are $\overset{x_i}{(1)}$ and $\overset{x_i}{(0)}$ which satisfy (3.14). Thus if $\alpha_i \sim \alpha_{\bar{i}}$ holds, then $\alpha = \alpha_i \cup \alpha_{\bar{i}}$ is completely monotonic.

S vectors of four augmented variables given to the lowest level are also written in Fig.3.9. and obviously any three of these four S vectors satisfy assumption (3.14), but for four augmented variables $\{\alpha_{ij}, \alpha_{i\bar{j}}, \alpha_{\bar{i}j}, \alpha_{\bar{i}\bar{j}}\}$ assumption (3.14) does not hold because

$$(1\ 1) + (0\ 0) = (1\ 0) + (0\ 1).$$

Other combinations are also possible, for example $\{\alpha_i, \alpha_{\bar{i}j}\}$ has S vectors $\overset{x_i}{(1)}$ and $\overset{x_i}{(0)}$, and this also satisfies (3.14).

Example 3.5: Consider function (3.10) given in Example 3.4. Applying Theorem 3.3 to the expansion diagram shown in Fig.3.7, the possible combinations of augmented variables are $\{\alpha_{12}, \alpha_{\bar{1}}\}$ and two or three of $\{\alpha_{12}, \alpha_{\bar{1}2}, \alpha_{1\bar{2}}\}$. Here we try the most effective one, i.e., $\{\alpha_{12}, \alpha_{\bar{1}2}, \alpha_{1\bar{2}}\}$. S vectors for these augmented variables are $\overset{x_1\ x_2}{(1, 1)}$, (0, 1), and (0, 0) which obviously satisfy

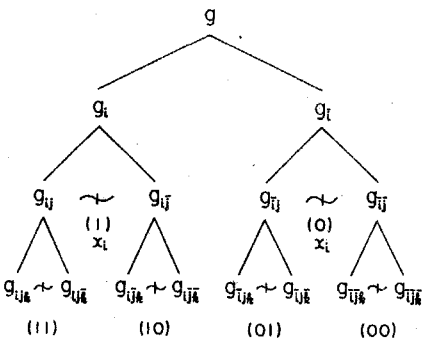


Fig.3.9. Illustration of augmented variables which can be united.

(3.14). Let γ_a be a $\gamma^{(j)}$ in (3.13) and correspond to α_a. Now if we select each γ as follows:

$$\gamma_{12} = x_4x_5x_3 \vee 0\bar{x}_3 = x_3x_4x_5$$
$$\gamma_{\bar{1}2} = x_4x_5x_3 \vee 0\bar{x}_3 = x_3x_4x_5$$
$$\gamma_{1\bar{2}} = 1x_3 \vee x_4\bar{x}_3 = x_3 \vee x_4,$$

then it is easily varified by the method developed previously that any two of these three functions are mutually monotonic, Therefore, the united function

$$\beta = \alpha_{12} \cup \alpha_{\bar{1}2} \cup \alpha_{1\bar{2}}$$
$$= x_1x_2\gamma_{12} \cup \bar{x}_1x_2\gamma_{\bar{1}2} \cup \bar{x}_1\bar{x}_2\gamma_{1\bar{2}}$$

is completely monotonic. This β is an incompletely specified function and expressed as follows:

$$\beta:\begin{cases}\beta^1 = x_1x_2(x_3x_4x_5) \vee \bar{x}_1x_2(x_3x_4x_5) \vee \bar{x}_1\bar{x}_2(x_3 \vee x_4)\\ \quad = x_2x_3x_4x_5 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1\bar{x}_2x_4\\ \beta^0 = x_1x_2(\overline{x_3x_4x_5}) \vee \bar{x}_1x_2(\overline{x_3x_4x_5}) \vee \bar{x}_1\bar{x}_2(\overline{x_3 \vee x_4})\\ \quad = x_2\bar{x}_3 \vee x_2\bar{x}_4 \vee x_2\bar{x}_5 \vee \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4.\end{cases} \qquad (3.16)$$

The realization by this β is shown in Fig.3.10. Note that three elements A, B, and C of Fig.3.8 are united into one in this example.

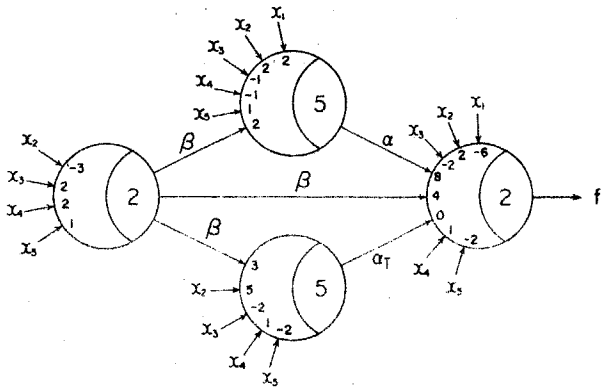


Fig.3.10. Realization of the functions in Example 3.4 by threshold elements with the aid of Uniting method.

It is sometimes possible to unite the augmented variables of each level into one, though it is not generally possible except at level 1.

In the above discussion, it is necessary to examine the mutual monotonicity of augmented variables. Next, the determination of their mutual monotonicity from the strucrure of corresponding non-mutually monotonic pairs

in the expansion diagram will be considered.

Consider two non-mutually monotonic pairs $f_{ai} \propto f_{\bar{a}i}$ and $f_{bi} \sim f_{\bar{b}i}$ in the same level. For the sake of simplicity, let these pairs be denoted as $p \propto q$ and $r \propto s$, respectively. Augmented variables given to these pairs be α and β, respecrively. To represent α and β, we again employ the notation of Fig.3.2, namely,

α:

$U(p)$	$U(q)$
$V(p)$	$V(q)$

β:

$U(r)$	$U(s)$
$V(r)$	$V(s)$

For example,

$$\alpha = \begin{array}{|c|c|}\hline 0 & 0 \\ \hline 1 & 0 \\ \hline \end{array}$$

implies $\alpha = \bar{p}x_i \vee 0\bar{x}_i$.

First we claim the following property for α and β to be mutually monotonic.

Theorem 3.6: Let t be one of two functions p and q, and u be one of two fucctions r and s. If for any t and u, both $t \propto u$ and $\bar{t} \propto u$ (i.e., $t \propto \bar{u}$) hold, then α and β derived from Theorem 3.1 always satisfy $\alpha \propto \beta$.

Proof: For each augmented variable of Theorem 3.1 represented by the above notation there is one column in which it assumes different values for U(*) and V(*) (see Fig.3.2). Now suppose, for example,

$$\alpha = \begin{array}{c|c} p & \\ \hline 1 & \\ \hline 0 & \end{array}, \qquad \beta = \begin{array}{c|c} & s \\ \hline & 0 \\ \hline & 1 \end{array},$$

then from the assumption of the theorem, p $\not\propto$ $\bar{s}$ holds, which implies the existence of four vectors:

$$u_1 \in U(p), \qquad u_2 \in U(\bar{s}) = V(s),$$
$$v_1 \in V(p), \qquad v_2 \in V(\bar{s}) = U(s),$$
$$u_1 + u_2 = v_1 + v_2.$$

If we write the x_i component to the left of these vectors, we obtain

$$(1, u_1) \in U(\alpha)$$
$$(1, v_1) \in V(\alpha)$$
$$(0, u_2) \in U(\beta)$$
$$(0, v_2) \in V(\beta)$$
$$(1, u_1) + (0, u_2) = (1, v_1) + (0, v_2),$$

and this is equivalent to $\alpha \not\sim \beta$. In other cases, it can be shown similarly that $\alpha \not\propto \beta$ holds. Q.E.D.

As a result, we cannot unite such α and β.

The converse of Theorem 3.6 holds in a sense and it is summarized in the next theorem.

Theorem 3.7: If the assumption of Theorem 3.6 does not hold, augmented variables which are shown in Table 3.1 satisfy Theorem 3.1; furthermore, $\alpha \sim \beta$ holds.

Proof: For simplicity, suppose $p \sim r$,

$$\alpha = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 0 \\ \hline \end{array}$$

and

$$\beta = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 0 \\ \hline \end{array} .$$

by putting $\varepsilon = 1$ and $\mu = 0$.

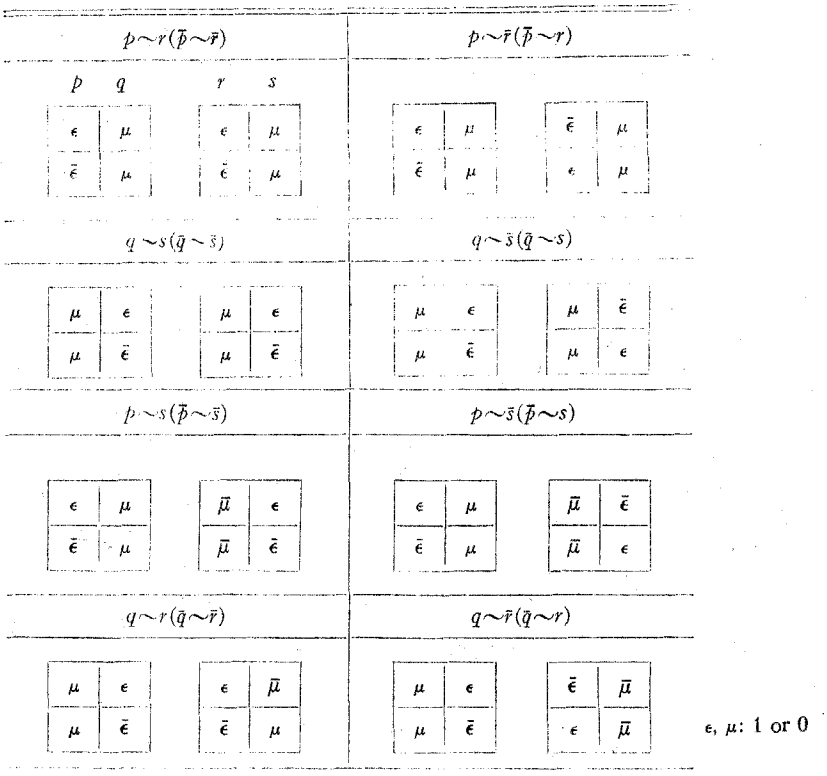


Table 3.1. Assignment of an augmented variable of Theorem 3.7

Other cases can be proved similarly. These are obviously the α and β permitted by Theorem 3.1. The x_i component of input vector which belongs to $U(\alpha)$ or $U(\beta)$ is necessarily 1. If $\alpha \not\sim \beta$, there must be four vectors

$$u_1 \in U(\alpha), \quad u_2 \in U(\beta)$$
$$v_1 \in V(\alpha), \quad v_2 \in V(\beta)$$
$$u_1 + u_2 = v_1 + v_2,$$

where the x_i components of u_1 and u_2 are both 1 and thus the x_i components of v_1 and v_2 are also 1. Then

$$u_1' \in U(p), \quad u_2' \in U(r)$$
$$v_1' \in V(p), \quad v_2' \in V(r)$$
$$u_1' + u_2' = v_1' + v_2',$$

where u_1' is u_1 from which the x_i component was deleted, etc. This implies, however, that $p \not\sim r$, which is a contradiction. Thus necessarily $\alpha \sim \beta$ holds.

Q.E.D.

By the use of Table 3.1 we can determine the augmented variables so as to satisfy the required properties, provided that we know the mutual monotonicity of each pair of functions such as p, q, r, and s.

For example, the augmented variable β in Example 3.5 is also illustrated under level 2 in Fig 3.7. It is not difficult to confirm that any two of these certainly satisfy Theorem 3.7, because $f_{123} \sim f_{\bar{1}23}$, $f_{\bar{1}23} \sim f_{\bar{1}\bar{2}\bar{3}}$ and $f_{123} \sim f_{\bar{1}\bar{2}\bar{3}}$ hold. Thus β in Example 3.5 could be determined rather automatically.

3.6 Elimination of Augmented Variables in the Upper Level

Sometimes it is possible to eliminate augmented variables which correspond to non-mutually monotonic pairs in the upper level by other augmented variables already obtained below them.

Example 3.6: Consider the augmented variable β in Example 3.4 given to the function (3.10) of Example 3.4. By this β, the augmented variable α_1 becomes unnecessary. Because if we put

$$f'_{\bar{1}2}(x_3, x_4, x_5, \beta) = f_{\bar{1}2}(x_3, x_4, x_5)$$
$$f'_{\bar{1}\bar{2}}(x_3, x_4, x_5, \beta) = f_{\bar{1}\bar{2}}(x_3, x_4, x_5).$$

then from the value of β shown in Fig.3.7,

$$\begin{aligned}
(f'_{\bar{1}2})^1 &= f_{\bar{1}23}\cdot x_3\cdot\beta \vee f_{\bar{1}2\bar{3}}\cdot\bar{x}_3\cdot\bar{\beta}\\
(f'_{\bar{1}2})^0 &= f_{\bar{1}23}\cdot x_3\cdot\bar{\beta} \vee f_{\bar{1}2\bar{3}}\cdot\bar{x}_3\cdot\bar{\beta}\\
(f'_{\bar{1}\bar{2}})^1 &= f_{\bar{1}\bar{2}3}\cdot x_3\cdot\beta \vee f_{\bar{1}\bar{2}\bar{3}}\cdot\bar{x}_3\cdot\beta\\
(f'_{\bar{1}\bar{2}})^0 &= f_{\bar{1}\bar{2}3}\cdot x_3\cdot\beta \vee f_{\bar{1}\bar{2}\bar{3}}\cdot\bar{x}_3\cdot\bar{\beta}.
\end{aligned} \tag{3.17}$$

Namely,

$$\begin{aligned}
(f'_{\bar{1}2})^1 &= x_3x_4x_5\beta \vee \bar{x}_3\bar{x}_5\bar{\beta}\\
(f'_{\bar{1}2})^0 &= (x_3\bar{x}_4 \vee x_3\bar{x}_5 \vee \bar{x}_3x_5)\bar{\beta}\\
(f'_{\bar{1}\bar{2}})^1 &= x_3\bar{x}_5\beta \vee \bar{x}_3x_4\beta\\
(f'_{\bar{1}\bar{2}})^0 &= x_3x_5\beta \vee \bar{x}_3\bar{x}_4\bar{\beta}.
\end{aligned} \tag{3.18}$$

Then by the earlier method in Chpter 2, we can show $f'_{12} \sim f'_{\bar{1}2}$. Consequently, the element in Fig.3.10 which produces α_1 is unnecessary and f is realized by the combination of three completely monotonic (threhold) functions.

In general, the possibility of eliminating augmented variables in such a way as shown in Example 3.6 must be examined. It is not easy to select augmented variables of lower levels so as to eliminate upper augmented variables. In special cases, however, this can also be done automatically.

Theorem 3.8: Consider part of an expansion diagram as shown in Fig.3.11, and let t be one of two functions p and q, and u be one of r and s. Then an augmented variable which is shown in Table 3.2 and given to both $p \not\sim q$ and $r \not\sim s$ ($p \sim q$ or $r \sim s$ are also possible) can remove the non-mutually monotonic pair $g_i \not\sim g_{\bar{i}}$ and therefore can eliminate the augmented variable assigned to it.

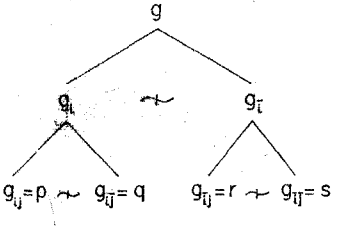
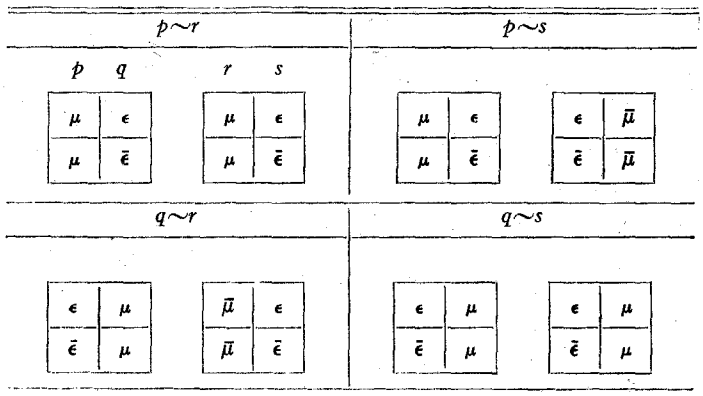


Fig. 3.11. Part of an expansion diagram.



ϵ, μ: 1 or 0

Table 3.2. Assignment of an augmented variable of Theorem 3.8.

Proof: Suppose p ∿ q and an augmented variable δ is such that

$$\delta: \left(\begin{array}{|c|c|} \hline p & q \\ \hline 0 & 1 \\ \hline 0 & 0 \\ \hline \end{array} \qquad \begin{array}{|c|c|} \hline r & s \\ \hline 0 & 1 \\ \hline 0 & 0 \\ \hline \end{array} \right) \tag{3.19}$$

from Table 3.2. Let g_i' be g_i augmented with the variable δ, etc., then the x_j and δ components of input vectors in $U(g_i')$ are one of (1 0) and (0 1), because

$$(g_i')^1 = p x_j \bar{\delta} \vee q \bar{x}_j \delta \tag{3.20}$$

holds. Similarly, the x_j and δ components corresponding to other sets are

$$
\begin{array}{lll}
 & x_j\ \delta & x_j\ \delta \\
V(g_i'): & (1\ \ 0), & (0\ \ 0) \\
U(g_i'): & (1\ \ 0), & (0\ \ 1) \\
V(g_{\bar{i}}'): & (1\ \ 0), & (0\ \ 0).
\end{array} \qquad (3.21)
$$

In this case, if $g_i' \not\propto g_{\bar{i}}'$ holds, there are four vectors

$$
\begin{aligned}
&u_1 \in U(g_i'), \qquad u_2 \in U(g_i') \\
&v_1 \in V(g_i'), \qquad v_2 \in V(g_{\bar{i}}') \\
&u_1 + u_2 = v_1 + v_2.
\end{aligned}
$$

However, if we consider (3.21), it is derived by examining all possible configurations that they must be written as follows:

$$
\begin{array}{ll}
 & x_j\ \delta \\
u_1 = & (1\ \ 0\cdots) \\
v_1 = & (1\ \ 0\cdots) \\
u_2 = & (1\ \ 0\cdots) \\
v_2 = & (1\ \ 0\cdots).
\end{array}
$$

This implies $p = g_{ij} \not\propto r = g_{\bar{i}j}$ and this is a contradiction. Other cases can be proven similarly.

Q.E.D.

It is obvious that δ removes non-mutually monotonic pairs $p \not\propto q$ and $r \not\propto s$ because it satisfies Theorem 3.1.

Note, however, that this theorem does not claim that δ is completely monotonic. For δ to be completely monotonic it must satisfy Theorem 3.7 at the same time.

Theorems 3.7 and 3.8 are compatible if and only if one of the following two cases occurs.

(1) p $\sim$ r and q $\sim$ s, then

$$\delta = \left(\begin{array}{cc|cc} p & q & r & s \\ \hline \epsilon & \mu & \epsilon & \mu \\ \bar{\epsilon} & \mu & \bar{\epsilon} & \mu \end{array} \right)$$

or

$$\left(\begin{array}{cc|cc} \mu & \epsilon & \mu & \epsilon \\ \mu & \bar{\epsilon} & \mu & \bar{\epsilon} \end{array} \right) . \qquad (3.22A)$$

(2) p $\sim$ s and q $\sim$ r, then

$$\delta = \left(\begin{array}{cc|cc} \epsilon & \mu & \bar{\mu} & \epsilon \\ \bar{\epsilon} & \mu & \bar{\mu} & \bar{\epsilon} \end{array} \right)$$

or

$$\left(\begin{array}{cc|cc} \mu & \epsilon & \epsilon & \bar{\mu} \\ \mu & \bar{\epsilon} & \bar{\epsilon} & \bar{\mu} \end{array} \right) . \qquad (3.22B)$$

where ε and μ are 1 or 0.

Note that the augmented variable β in Examples 3.5 and 3.6 satisfies the second condition for the pair of functions $f_{\tilde{1}2} \propto f_{\tilde{1}\tilde{2}}$ and therefore it was unnecessary to examine the mutual monotonicity of $f'_{\tilde{1}2} \sim f'_{\tilde{1}\tilde{2}}$ as performed in Example 3.6.

When Theorems 3.7 and 3.8 are not compatible, Theorem 3.7 seems preferable, though it is possible to realize δ by a combination of completely monotonic functions by

employing again the previous techniques after realizing δ by Theorem 3.8.

3.7 Other Simplifications

Our techniques may be carried out easily and rather systematically since they depend on the mutual monotonicity of expanded functions and the mutual monotonicity can be examined easily by algebraic manipulation. However, augmented variables other than those obtained from Theorem 3.1 are also possible and sometimes they may reduce the number of required elements further.

Hopcroft and Mattson provided us with an interesting method[45] which first tabulates all the 2-summable vectors and then searches all possible augmented variables which remove all the 2-summabilities. Although their method can yield an optimal one if we exhaust all possible assignments of augmented variables, it is very complicated and almost impossible to implement by hand computation even for functions of few variables.

In a sense, our method is an effective assignment of augmented variables which removes all 2-summabilities among vectors. Therefore, to try augmented variables other than those obtained from Theorem 3.1 will eventually result in the Hopcroft anf Mattson method.

Thus, a reasonable method seems to construct a network which can realize a given function by the previous our method and then to refine the network by employing other assignments of augmented variables if possible.

It should be mentioned here about the approach proposed by Lewis and Coates[15][62]. Their method is also based on the manipulation of the expansion diagram (the function tree in their term). The algorithm starts

from the bottom of the expansion diagram, searching a threshold gate configuration which realizes the given function. Their idea has, roughly speaking, some similarity with the threshold function version of our algorithm (see Section 3.8). However, it involves the repetition of trial and error processes to find sumultaneously realizable threshold functions and their structures. For this reason it may not be practical except for small functions or functions with some special structure.

3.8 Extension to Threshold Logic

The realization by threshold (1-realizable) functions is of primal significance and our method is only an approximate solution to this problem since there exist completely monotonic functions which are not threshold functions.

In Chapter 2, however, it has been shown that the theory of completely monotonic functions can be extended to threshold functions if we replace the mutual monotonicity with the isobaricity. In fact, all theorems in this chapter except Theorem 3.5* are also true if we replace the complete monotonicity with the 1-realizability and the mutual monotonicity with the isobaricity.

Therefore, the method of compound synthesis in this chapter may also be extended to threshold logic. Note, however, that the determination of the isobaricity is not so simple as that of the mutual monotonicity, and this method cannot be applied easily.

However, when we consider completely specified functions, the difference between completely monotonic functions and threshold functions is not a great impediment because for functions of up to eight variables complete monotonicity is equivalent to 1-realizability.

---

*Theorem 3.5 can be extended to threshold logic if $k = 2$. This is, in a sense, a restatement of Theorem 2.12 in the previous chapter.

Functions of up to seven variables are isobaric if and only if they are mutually monotonic (Theorem 2.13).

Then, completely monotonic functions which are not 1-realizable can occur in the previous procedure only when there are mutually monotonic pairs but not isobaric; it is possible only if two functions of more than seven variables are compared, or if Theorem 3.5 yields non-threshold functions by uniting several augmented variables.

Note that logic elements in the resulting network may realize incompletely specified functions even though functions in the expansion diagram are all completely specified, since augmented variables are added. However, the present argument is valid if original functions in the expansion diagram are completely specified,

3.9 Summary of the Procedure

Here we shall summarize the method of realizing an arbitrary function by a combination of threshold functions. Although this procedure does not always yield an optimal one, the results are always considerably economical in the author's experience. Since this method is somewhat heuristic, this may be suitable to hand computation, though it is possible to program it for a computer to some extent.

(1) Given a function f, expand f and obtain its expansion diagram. If there is no non-mutually monotonic pair in it, f is completely monotonic. If there are more than one non-mutually monotonic pairs, try each sequence of expansion variables so as to reduce the number of non-mutually monotonic pairs as much as possible.

(2) Examine the mutual monotonicity of any pair of functions in the same level and, if necessary, functions in different levels, and try the following procedure so that the resulting number of augmented variables may become as small as possible.

  (a) Unite the augmented variables in the same level (Theorems 3.4, 5, 6, and 7).

  (b) Eliminate upper augmented variables if possible (Theorem 3.8 and (3.22)).

  (c) Try to unite augmented variables in different levels (Theorems 3.3, 4, and 5).

(d) If there are augmented variables other than those obtainable by Theorem 3.1, which reduce the number of augmented variables, adopt them

(e) Examine if all augmented variables obtained are actually necessary (Example 3.6).

(3) Find the weight vector and threshold value of each function obtained above by means of suitable synthesis procedure of a single threshold element such as the linear programming method. If all required functions are 1-realizable, then it is a solution of this problem. If some of them are not 1-realizable, examine if there are non-isobaric pairs which are mutually monotonic and if Theorem 3.5 yields completely monotonic functions which are not 1-realizable in the uniting procedure. Then apply to them the threshold version of the above procedure.

By this approach, we may obtain a considerably economical realization of a given function. Note that all steps in the procedure are not always necessary and after acquiring some amount of exparience it may be performed rather intuitively.

3.10 Examples

Example 3.7: Consider a completely specified function

$$f = x_1x_2x_3x_4 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_2x_3\bar{x}_4 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3x_4.$$

First expand this function by x_3 and then by x_2. The resulting expansion diagram is shown in Fig.3.12. Obviously, functions in level 2 are all completely monotonic and there are three non-mutually monotonic

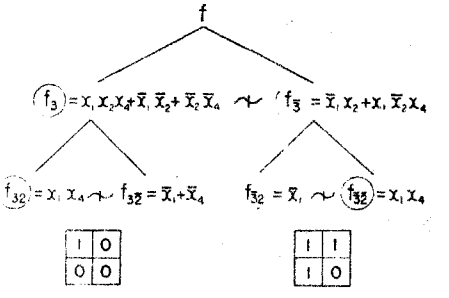


Fig.3.12. Expansion diagram of the function in Example 3.7.

pairs. Let augmented variables given to them be α, α_3, and $\alpha_{\bar{3}}$, respectively. Since $f_{32} \sim f_{\bar{3}\bar{2}}$ and $f_{3\bar{2}} \sim f_{\bar{3}2}$ are obtained immediately, we can apply Theorem 3.7 and Theorem 3.8 so that we may unite two augmented variables α_3, $\alpha_{\bar{3}}$ in level 2 and at the same time eliminate the augmented variable α in level 3. The assignment of the augmented variable is shown in Fig.3.12 under level 2. Of course it is obtained directly from (3.22B):

$$\beta = \alpha_3 \cup \alpha_{\bar{3}} = (x_1x_4x_2 \vee 0\cdot\bar{x}_2)x_3 \vee (1\cdot x_2 \vee x_1x_4\bar{x}_2)\bar{x}_3$$
$$= x_1x_2x_4 \vee x_2\bar{x}_3 \vee x_1\bar{x}_3x_4.$$

The resulting configuration of threshold elements can be determined as follows. First consider separately the above three augmented variables designated in Fig.3.12. The essential function to each augmented variable is circled as shown in Fig.3.12. Suppose an essential function in level 3 be f_3. Then applying the procedure in Section 3.5, the resulting network is as shown in Fig.3.13(a). Finally, corresponding to the union of α_3 and $\alpha_{\bar{3}}$, and the elimination of α, elements A and B are

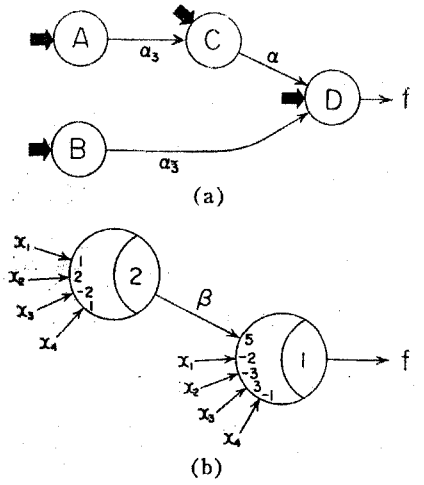


Fig. 3.13. Realization of the function in Example 3.7.

united into one and the element C is eiminated. Fig.3.13(b) shows a final realization which requires the minimal number of elements.

This function is used in Hopcroft and Mattson[45] to exhibit the effectiveness of their method. However, by our method, though it needs several trials to obtain an adequate expansion diagram, the synthesis procedure is simple and direct as seen from the above.

Example 3.8: Parity functions. The class of parity functions is the only class which yields the expansion diagram in which all pairs of functions in levels 1, 2,..., n-1 are non-mutually monotonic, for any sequence of expansion variables. Thus, in this sense, the parity function is opposite to the completely monotonic function.

Consider an even parity function of four variables:

$$f = x_1\bar{x}_2x_3x_4 \vee x_1x_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2x_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_3x_4 \vee \bar{x}_1x_2x_3\bar{x}_4 \vee \bar{x}_1x_2\bar{x}_3x_4 \vee \bar{x}_1\bar{x}_2x_3x_4 \vee \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4. \quad (3.23)$$

The expansion diagram of f is shown in Fig.3.14. Obviously, in level 1, $f_{123} \sim f_{1\bar{2}\bar{3}}$ and $f_{12\bar{3}} \sim f_{1\bar{2}3}$ hold; therefore, the union of augmented variables α_{12} and $\alpha_{1\bar{2}}$ which satisfy (3.22B) is completely monotonic and, moreover, can eliminate the augmented variable given to $f_{12} \not\sim f_{1\bar{2}}$. In fact, if we put

$$\beta_1 = \alpha_{12} \cup \alpha_{1\bar{2}}$$

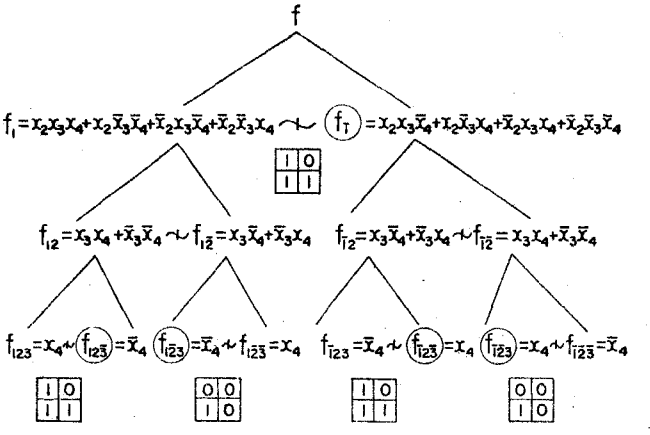


Fig.3.14. Expansion diagram of a parity function (Example 3.8).

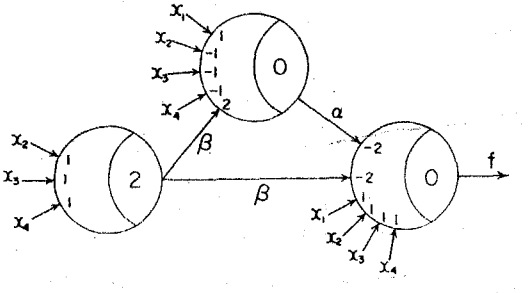


Fig.3.15. Realization of a parity function (Example 3.8).

then

$$\beta_1^1 = x_1(x_2x_3 \vee x_3x_4 \vee x_4x_2)$$
$$\beta_1^0 = x_1(\bar{x}_2\bar{x}_3 \vee \bar{x}_3\bar{x}_4 \vee \bar{x}_4\bar{x}_2),$$

which is clearly 1-realizable. From the symmetry of the parity function

$$\beta_{\bar{1}} = \alpha_{\bar{1}2} \cup \alpha_{\bar{1}\bar{2}}$$

can also be completely monotonic; moreover, a completely specified function

$$\beta = \beta_1 \cup \beta_{\bar{1}}$$

can be expressed as

$$\beta = x_2x_3 \vee x_3x_4 \vee x_4x_2, \qquad (3.24)$$

which is a simple three-input majority function. Thus only a single augmented variable β is required for all non-mutually monotonic pairs in levels 1 and 2. In order to realize f, we need one more augmented variable α for $f_1 \propto f_{\bar{1}}$. The resulting network is shown in Fig.3.15.

Repeating this procedure, a parity function of n variables can be realized by [n/2]+1 threshold elements. Though this is not minimal, it may be said to be rather economical since $[\log_2 n] + 1$ is known to be minimal[57].

Example 3.9: Consider an incompletely specified function f shown in Fig.3.16, in which there is only one non-

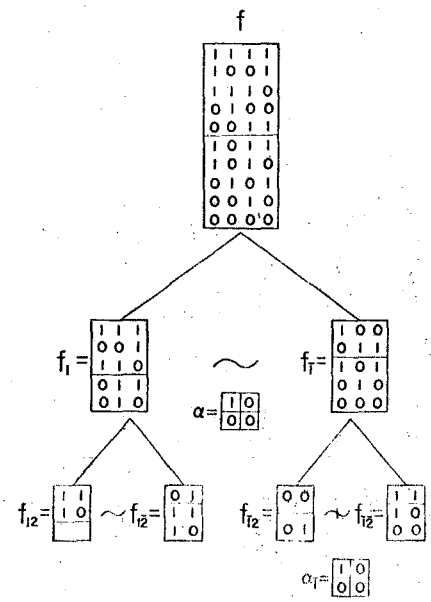


Fig.3.16. Expansion diagram of the function in Example 3.9.

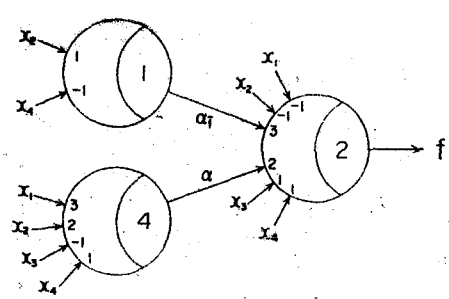


Fig.3.17. Realization of the function in Example 3.9.

mutually monotonic pair. However,

$$f'(x_1, x_2, x_3, x_4, \alpha_{\bar{1}}) = f(x_1, x_2, x_3, x_4)$$

is not 1-realizable because f_1 and $f_{\bar{1}}$ are mutually monotonic but not isobaric. Namely,

$$\begin{aligned}
&\boldsymbol{u}_1 = (0\ 0\ 1),\ \boldsymbol{u}_2 = (1\ 1\ 0) \in U(f_1)\\
&\boldsymbol{v}_1 = (0\ 1\ 1),\ \boldsymbol{v}_2 = (0\ 1\ 0) \in V(f_1)\\
&\boldsymbol{u}_3 = (0\ 1\ 1) \in U(f_{\bar{1}})\\
&\boldsymbol{v}_3 = (1\ 0\ 1) \in V(f_{\bar{1}})\\
&\boldsymbol{u}_1 + \boldsymbol{u}_2 + \boldsymbol{u}_3 = \boldsymbol{v}_1 + \boldsymbol{v}_2 + \boldsymbol{v}_3
\end{aligned}$$

hold. Thus another augmented variable α is necessary and by these two augmented variables f can be realized as in Fig.3.17.

3.11 Multiple-Output Problems

In practical applications, syntheses of multiple-output networks are very important. The synthesis procedure of a single-output function developed in the previous sections can be also easily extended to this problem. In this case, however, it is profitable to consider in addition the following properties.

(1) The same augmented variable can be given to different functions simultaneously.

(2) It is possible to unite augmented variables for different functions.

(3) A realized function can be used again as an augmented variable.

Now, the problem is how we can reduce the number of augmented variables as a whole by adding these properties to the previous method.

We consider only loop-free realizations of given functions and we have to pay attention so that no loop may result.

Example 3.10: Realize the following four functions:

$$
\begin{aligned}
e &= x_1x_2 \vee x_2x_3 \vee x_3x_1 \\
f &= x_1x_2\bar{x}_3 \vee x_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3 \\
g &= \bar{x}_1\bar{x}_2\bar{x}_3 \vee x_1x_2x_3 \vee x_1\bar{x}_2\bar{x}_3 \\
h &= x_1x_2 \vee x_1\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3.
\end{aligned}
\tag{3.25}
$$

Expansion diagrams of these functions are shown in Fig.3.18. Obviously e is a completely monotonic (threshold) function. We have to remove six non-mutually monotonic pairs in order to realize these four functions.
First, f is a parity function and according to Example 3.8, all non-mutually monotonic pairs of f are removed by an augmented variable

$$\alpha = x_1x_2 \vee x_2x_3 \vee x_3x_1,$$

which is the function e itself. Next, e = α can also remove the pair $g_{12} \nsim g_{1\bar{2}}$ because $f_{12} = \bar{g}_{12}$ and $f_{1\bar{2}} = \bar{g}_{1\bar{2}}$ hold. Then only two pairs $g_1 \nsim g_{\bar{1}}$ and $h_1 \nsim h_{\bar{1}}$ remain and

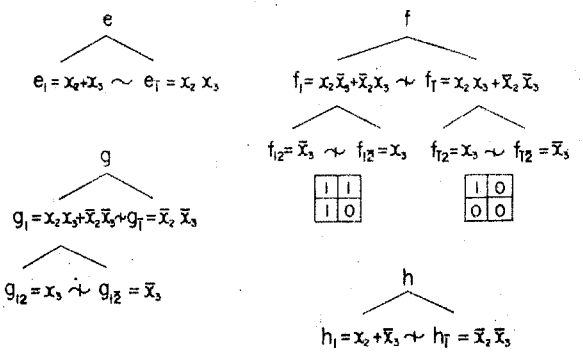


Fig.3.18. Expansion diagrams of functions in Example 3.10.

a single augmented variable such as

$$\beta = 0 \cdot x_1 \vee \bar{x}_2\bar{x}_3x_1 = \bar{x}_1\bar{x}_2\bar{x}_3$$

can remove both of them because both pairs are defined

on the same input vectors and, moreover $g_{\bar{l}} = h_{\bar{l}}$ holds. Consequently, Fig. 3.19 simultaneously realizes all four functions e, f, g, and h.

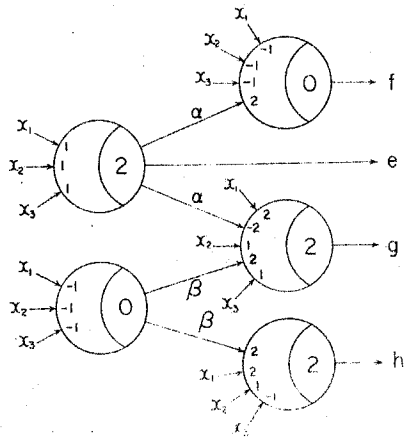


Fig.3.19. Multiple-output realization of functions in Example 3.10.

3.12 Conclusion

The concept of mutual monotonicity was shown to be useful in realizing an arbitrary function by a combination of completely monotonic functions. Although this method does not always yield an optimal or minimal network, the result is always considerably economical in the author's experience.

However, two problems remain. One is that the weight vector and threshold value of each element are not obtained from this procedure, and the other is that there are completely monotonic functions which are not 1-realizable.

To obtain the weight vector and threshold value many effective procedures have been proposed such as the linear programming method[69][72][76], the learning method[82], and the iteration method[20], if the function is known to be 1-realizable.

The latter problem is solved completely if isobaricity is employed instead of mutual monotonicity. Therefore, there is now an acute need to devise a simple procedure of examining the isobaricity relation.

Since the mutual monotonicity can be examined by algebraic manipulation and it is carried out easily when completely specified functions are concerned, our method works well for the realization of completely specified functions, whereas integer programming methods[11][9][80] etc., are particularly powerful for sparsely specified functions. When functions are completely specified, the difference between completely

monotonic functions and threshold functions is also not a serious impediment bacause such functions can exist only when the number of variables exceeds eight.

It is mentioned here that a different type of networks of threshold gates has received attention[3][38][53][54][90][110]. The model consists of n threshold elements mutually interconnected with no external variable fed to the network. The sequential bahavior of this autonomous network exhibits many intersting features. For this type of network, it would be also possible to apply the similar theory of complete monotonic functions.

Chapter 4. Network Synthesis Using Negative Functions

4.1 Introduction

As briefly reviewed in Section 1.4, the recent development of MOS gates in integrated circuit techonology renders each gate represent a rather complex negative function. A typical MOS gate shown in Fig.4.1 realizes [111] the negative function:

$$g = \overline{x_{11}x_{12}\cdots x_{1k_1} \vee \cdots \vee x_{s1}x_{s2}\cdots x_{sk_s}}.$$

ϕ_1 and ϕ_2 in Fig.4.1 are clock signals supplied from the external clock circuit.

Assuming that each gate can represent any negative function, there could be a way of realizing a rather complex function f by using a small number of negative functions.

As an example, consider a function of three variables x_1, x_2, x_3.

$$f = \bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_3 \vee x_1x_2x_3. \qquad (4.1)$$

One obvious design procedure of f with negative functions is first to realize three complemented variables $\bar{x}_1$, $\bar{x}_2$, $\bar{x}_3$, using three gates respectively and then to realize f by regarding f as a negative function of x_1, x_2, x_3, $\bar{x}_1$, $\bar{x}_2$, $\bar{x}_3$. This is illustrated in Fig.4.2, in which each gate g_i represents a negative function.

This synthesis procedure can be generalized to any

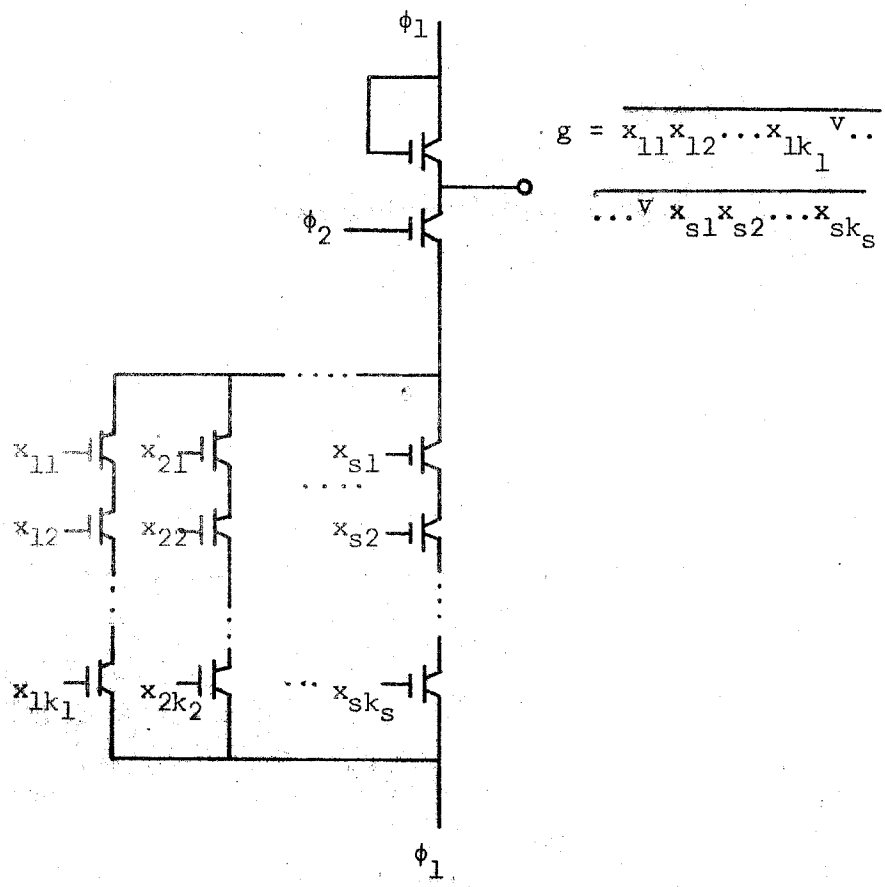


Fig.4.1. A typical MOS gate.

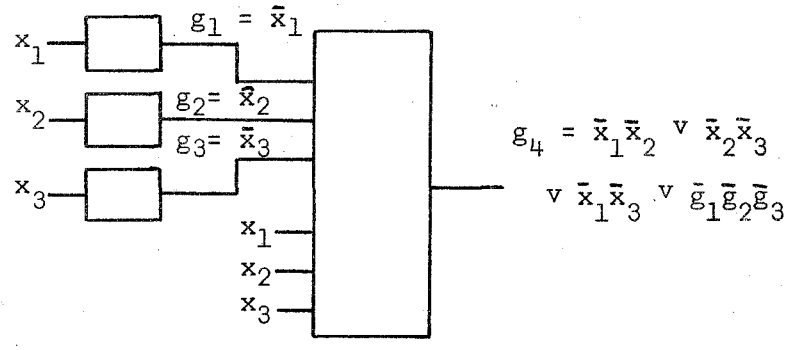


Fig.4.2. Straight-forward realization of $f = \bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_3 \vee x_1x_2x_3$.

function, indicating that any function of n variables can be realized with at most n+1 gates of negative functions within two levels.

The above network, however, can be simplified. A minimal network which is obtained by the method developed in this chapter is shown in Fig.4.3.

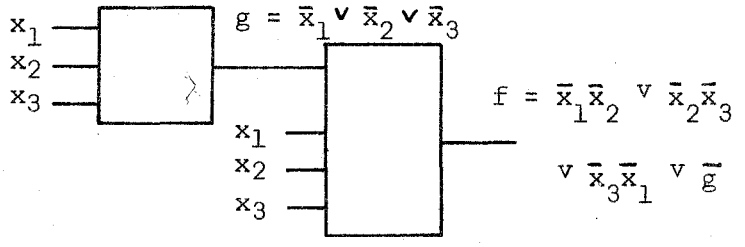


Fig.4.3 Minimum realization of $f = \bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_3 \vee x_1x_2x_3$.

Here, we mean by a minimum network a network with the minimum number of gates of negative functions under the conditions that;

(1) The given function f is realized within two levels (the level which consists of gates connected to the output gate is called the "first level" and the level of the output gate is the "second level").

(2) No fan-ins restriction on each gate is imposed.

We will not minimize the number of connections in the network except some limited cases discussed elswhere[50].

Although this minimization criterion may not exactly reflect the engineering requirements of the integrated circuit designers, this seems to be only criterion at present, which can be theoretically treated. It is to be explored whether or not a minimum network obtained under the above criterion would deviate much from those obtained under different minimality criteria. But we hope that the theory presented here would be a basis for further development.

4.2 Negative Functions

This section will define negative functions in terms of truth table. This will be a basis of the minimization procedure developed in this chapter.

Let us first consider a completely specified function. A wellknown definition of a negative function[1][76] is that a function is negative if and only if it has a disjunctive form with only complemented (i.e., negative) literals. For example, $\bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_3$ is negative while $\bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_3 \vee x_1x_2x_3$ is not. Since our procedure will be based on the manipulation of a truth table rather than Boolean expression, the following equivalent definition, which is also wellknown[1][76][98] would be more useful.

Definition 4.1: A completely specified function h of m variables $y_1,\ldots, y_m$ is negative if and only if there is no pair of input vectors $y^{(j_1)}$ and $y^{(j_2)}$ such that*:

$$y^{(j_1)} > y^{(j_2)} \qquad (4.2)$$

and

$$h(y^{(j_1)}) = 1$$

$$h(y^{(j_2)}) = 0, \qquad (4.3)$$

---

*Here we exclude from our consideration the contradictory situation such that

$$y^{(j_1)} = y^{(j_2)} \quad \text{and } h(y^{(j_1)}) = 1,\ h(y^{(j_2)}) = 0.$$

This will never occur in our procedure.

where $y^{(j_1)} > y^{(j_2)}$ is defined as

$$y_i^{(j_1)} \geq y_i^{(j_2)} \quad \text{for every } i = 1, 2, \ldots, m,$$

and

$$y_i^{(j_1)} > y_i^{(j_2)} \quad \text{for at least one } i.$$

As an example, the function f of variables x_1, x_2, x_3 shown in Table 4.1 (ignore the columns g_1, g_2, g_3 for a while) is not negative because there exists a pair such that

$$x^{(8)} = (111) > x^{(4)} = (011)$$

and

$$f(111) = 1, \; f(011) = 0.$$

This function is the same function as that discussed in Fig.4.1 and Fig.4.2.

	f	x_1	x_2	x_3	g_1	g_2	g_3
(1)	1	0	0	0	1	1	1
(2)	1	0	0	1	1	1	
(3)	1	0	1	0	1		1
(4)	0	0	1	1	1		
(5)	1	1	0	0		1	1
(6)	0	1	0	1		1	
(7)	0	1	1	0			1
(8)	1	1	1	1	0	0	0

Table 4.1. Truth table of $f = \bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_3 \vee x_1x_2x_3$.

An outline of our procedure which realizes a given general (not necessarily negative) function by means of a combination of negative functions is now given. First we augment the truth table by several columns (illustrated as columns g_1, g_2, g_3 in Table 4.1) so that the given function f may be considered as a negative function if those new columns together with columns $x_1,..,x_n$ are regarded as variables of f. According to our procedure, each of these new columns represents a negative function of variables $x_1, x_2,..., x_n$. Therefore, from this augmented table, we can construct a network of negative functions in which each gate in the first level corresponds to an augmented column and the gate in the second level, to which the outputs of gates in the first level as well as the variables $x_1, x_2,..., x_n$ are fed, realizes the function f (see Fig.4.4). In general, the augmented columns are incompletely specified, i.e., some entries of g_1, g_2, g_3 are left blank as seen in Table 4.1. Generation of augmented columns will be discussed later in detail.

Throughout this chapter, let us assume that a function is generally incompletely specified.

As observed in Table 4.1, an incompletely specified function is possibly not specified in some of the output values and also in entries in some input vectors. For example, the output values of $g_1(x_1, x_2, x_3)$ in Table 4.1 are not specified for input vectors (100), (101) and (110). Also we consider f as a function of input vectors $(x_1, x_2, x_3, g_1, g_2, g_3)$, then some of input vectors themselves are not completely specified. For example,

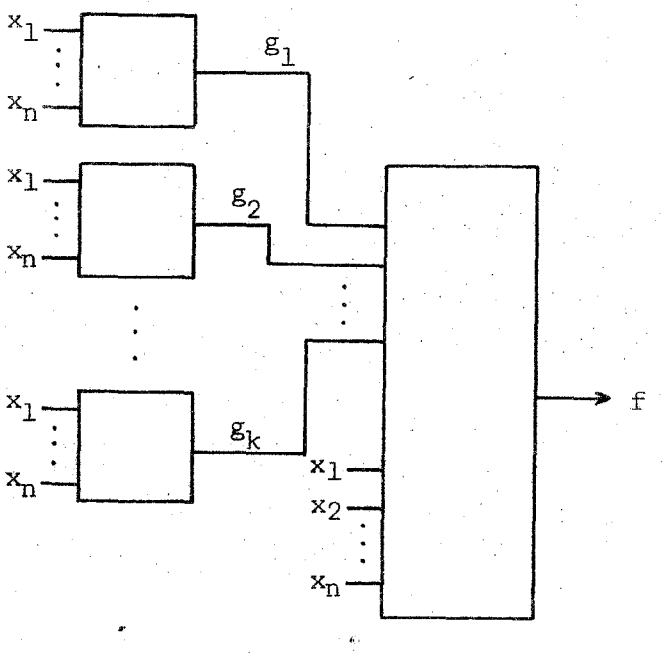


Fig.4.4. General scheme of realization with negative functions.

the second row of g_3 in Table 4.1 is not specified. In this case, we interprete that the output values of f assume the values indicated in the column f, regardless of the assignments of unspecified entries. For example, the second row of Table 4.1 shows that the actual output value of f is 1, no matter whether the input vector is (001110) or (001111), i.e., regardless of the value in column g_3 which is blank in Table 4.1.

Now let us introduce a definition.

Definition 4.2: A column c_i is said negative with respect to c_{i_1}, c_{i_2},...,c_{i_s} if ond only if for every pair of

entries $c_i^{(j_1)}$ and $c_i^{(j_2)}$ of c_i such that $c_i^{(j_1)} = 1$ and $c_i^{(j_2)} = 0$, there exists a subscript ℓ $(1 \leq \ell \leq s)$ which satisfies

$$c_{i_\ell}^{(j_1)} = 0$$

and (4.4)

$$c_{i_\ell}^{(j_2)} = 1 .$$

Consider the column f of Table 4.1. The column f is not negative with respect to x_1, x_2, and x_3 because, for the pair of the eighth and the fourth entries, $f^{(8)} = 1$ and $f^{(4)} = 0$, there is no subscript k $(1 \leq k \leq 3)$ such that $x_k^{(8)} = 0$ and $x_k^{(4)} = 1$. If x_1, x_2, x_3, g_1, g_2, g_3 are considered as variables of f, however, then f is negative as easily verified. Note that the definition of the negativeness of a column is valid no matter whether related columns are completely specified or not.

In the following, first we will prove that two concepts of definitions 4.1 and 4.2, the negativeness of a function and that of a column, are equivalent if a completely specified case is concerned (Theorem 4.1). Then this equivalence will be extended to the incompletely specified case (Theorem 4.2). With those arguments, all the discussions throughout the following sections will be done in terms of the negativeness of columns.

The next definition of a negative function stated as Theorem 4.1 is a variant of Aker's theorem[2] which gave a necessary and sufficient condition for a function being a self-dual positive function. A comprehensive

discussion is also found in [76].

Theorem 4.1: A completely specified function h of m variables $y_1, y_2, \ldots, y_m$ is negative if and only if the corresponding column h in the truth table is negative with respect to columns $y_1, y_2, \ldots, y_m$.

Proof: Let us prove that the condition for the column h to be negative with respect to columns $y_1, \ldots, y_m$ (Definition 4.2) is equivalent to the condition that there is no pair of input vectors of h, $y^{(j_1)}$ and $y^{(j_2)}$ such that $y^{(j_1)} > y^{(j_2)}$ and $h(y^{(j_1)}) = 1$, $h(y^{(j_2)}) = 0$ (Definition 4.1). Now note that $y^{(j_1)} > y^{(j_2)}$ is equivalent to non-existence* of any subscript k such that $y_k^{(j_1)} = 0$ and $y_k^{(j_2)} = 1$. Therefore, if for every pair of entries, $h(y^{(j_1)}) = 1$ and $h(y^{(j_2)}) = 0$, there exists a subscript k such that $y_k^{(j_1)} = 0$ and $y_k^{(j_2)} = 1$, it implies that there is no pair of vectors $y^{(j_1)}$ and $y^{(j_2)}$ such that $y^{(j_1)} > y^{(j_2)}$ and $h(y^{(j_1)}) = 1$, $h(y^{(j_2)}) = 0$, and vice versa.

Q.E.D.

In order to extend Theorem 4.1 to incompletely specified functions, the next definition is introduced.

Definition 4.3: A completion of h is a completely specified function obtained from h by specifying all the unspecified output values of h. If a resultant completely specified function is negative, it is a negative completion.

*$\overline{y^{(j_1)}} \neq y^{(j_2)}$ is assumed for every pair of $j_1 \neq j_2$.

Theorem 4.2: An incompletely specified function h of m variables $y_1, y_2, \ldots, y_m$ has a negative completion if and only if the column h in a truth table is negative with respect to columns $y_1, \ldots, y_m$.

Proof: The only-if part is trivial.

Now, consider an incompletely specified column h which is negative with respect to $y_1, \ldots, y_m$. First, if there are blank entries in columns $y_1, \ldots, y_m$, expand the table by assigning 0 and 1 to every blank entry, by multiplexing each row. (i.e., if a row is (01--1) where - denotes blank entries, replace this row by (01001), (01011), (01101), (01111).) As a result, the table has 2^m rows. The entries of the column h in the multiplexed rows of the new table are set to the same values as the original entries. This process takes care of input vectors which are incompletely specified. The column h in the new table is obviously still negative.

Next consider the case in which the entries in column h is incompletely specified. Denote the sets of true vactors and false vectors of h as follows:

$$A = \{ y^{(j)} \mid h(y^{(j)}) = 1 \}$$

$$B = \{ y^{(j)} \mid h(y^{(j)}) = 0 \}. \quad (4.5)$$

There is no pair of vectors $y^{(j_1)}$ and $y^{(j_2)}$ such that

$$y^{(j_1)} > y^{(j_2)} \quad \text{and} \quad y^{(j_1)} \in A, \quad y^{(j_2)} \in B$$

from the negativeness of column h. Then pick up a vector

$y^{(k)}$ for which the value $h(y^{(k)})$ is not specified yet. $y^{(k)}$ satisfies one of the following three cases:

(1) There exists a vector $y^{(p)}$ such that

$$y^{(p)} \in A \text{ and } y^{(p)} > y^{(k)}. \qquad (4.6)$$

(2) There exists a vector $y^{(q)}$ such that

$$y^{(q)} \in B \text{ and } y^{(q)} < y^{(k)}.$$

(3) None of (1) and (2) holds.

If case (1) occurs, then $h(y^{(k)})$ is specified to 1. Then the resulting function h is characterized by its set of true vectors and set of false vectors;

$$A' = A \cup \{y^{(k)}\}$$

and

$$B' = B$$

The new column h' is still negative with respect to columns $y_1, y_2, \ldots, y_m$, because, otherwise the existence of vectors such that

$$y^{(k)} > y^{(j)}, \quad y^{(k)} \in A', \quad y^{(j)} \in B',$$

leads to a contradiction, i.e.,

$$y^{(p)} > y^{(j)}, \quad y^{(p)} \in A' \quad y^{(j)} \in B',$$

from condition (4.6).

If (2) is the case, then $h(y^{(k)}) = 0$ is assigned.

The proof is similarly obtained.

Finally, if (3) is the case, either 0 or 1 is assigned to $h(y^{(k)})$. Either case still keeps the column negative, even though one more entry is specified. This is because the non-negativeness of h' means the existence of

$$y^{(k)} > y^{(j)}, \quad y^{(k)} \in A', \quad y^{(j)} \in B'$$

(Here assumed $h'(y^{(k)}) = 1$. The case of $h'(y^{(k)}) = 0$ will be treated similarly.) which is a contradiction since this $y^{(k)}$ satisfies the above case (2).

The assignment to the next unspecified entry of h' can be performed similarly. Repeating this process, we will eventually obtain a negative completion of h.

Q.E.D.

As a result of Theorem 4.1 and 4.2, we learned that the negativeness of a column h with respect to columns $y_1, \ldots, y_m$ means that function h represented by column h in the truth table is a negative function (or has a negative completion) of variables $y_1, \ldots, y_m$. Thus the negativeness of completely specified function has been extended to the case of incompletely specified function.

Since the completion procedure described in the proof of Theorem 4.2 does not always yield the unique negative completion, any negative completion will be assumed in the following discussion because we are concerned with only negativeness of each function.

4.3 Realization of a given function by negative functions

If a given function f is not negative, more than one gate is needed to realize f with negative functions. As outlined in Section 4.2, our algorithm for the optimal metwork is based on the table manipulation. It first augments the given truth table with columns $g_1, g_2, \ldots, g_k$ such that:

(1) The column f is negative with respect to $x_1, x_2, \ldots, x_n, g_1, \ldots, g_k$.

(2) The columns g_i, i = 1, 2, ..., k, are negative with respect to $x_1, x_2, \ldots, x_n$.

Provided that those columns $g_1, \ldots, g_k$ with the above properties are obtained, the corresponding network for f with negative functions is shown in Fig.4.4. Each gate in the first level corresponds to one of g_i's, and the gate in the second level realizes f as a function of $x_1, x_2, \ldots, x_n, g_1, \ldots, g_k$.

In this section, we will discuss how to obtain g_i which satisfy the above requirements. The minimization of the number of gates will be discussed in the subsequent sections.

Definition 4.4: A pair of entries $c_i^{(j_1)} = 1$ and $c_i^{(j_2)} = 0$ in a column c_i is unallowable with respect to c_{i_1}, $c_{i_2}, \ldots, c_{i_s}$ if and only if there is no subscript ℓ ($1 \leq \ell \leq s$) such that

$$c_{i_\ell}^{(j_1)} = 0 \text{ and } c_{i_\ell}^{(j_2)} = 1. \qquad (4.7)$$

If there exists a subscript ℓ which satisfies (4.7), $c_i^{(j_1)}$ and $c_i^{(j_2)}$ is said <u>allowable due to</u> c_{i_ℓ}.

Note that the column c_i is negative with respect to columns $c_{i_1}, c_{i_2}, \ldots, c_{i_s}$ if and only if any pair of entries of c_i such that $c_i^{(j_1)} = 1$ and $c_i^{(j_2)} = 0$ is allowable.

Now let us form the <u>negativity table</u> of function f of n variables $x_1, x_2, \ldots, x_n$. The negativity table is constructed according to the following procedure:

(1) Exhaust all unallowable pairs of column f with respect to columns $x_1, \ldots, x_n$, in the truth table of f.

(2) For each unallowble pair of entries, $f(X^{(j_1)}) = 1$ and $f(X^{(j_2)}) = 0$, add a new column e_j whose entries are $e_j^{(j_1)} = 0$, $e_j^{(j_2)} = 1$ and blank in all other entries. (i.e., e_j has exactly two entries which are not blank.) e_j is called a <u>primitive additional column</u> of f.

For example, the function f shown in Table 4.1 has three unallowable pairs of entries of f, as easily seen. For these three unallowable pairs, three columns e_1, e_2, e_3 are added to the truth table. The negativity table of f which results is shown in Table 4.2.

Because of the way each e_j being generated, the next theorem is obvious.

<u>Theorem</u> 4.3: Let $e_1, e_2, \ldots, e_t$ be all the primitive additional columns generated for the negativity table

f	x_1	x_2	x_3	e_1	e_2	e_3
1	0	0	0			
1	0	0	1			
1	0	1	0			
0	0	1	1	1		
1	1	0	0			
0	1	0	1		1	
0	1	1	0			1
1	1	1	1	0	0	0

Table 4.2. Negativity table of $f = \bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_3 \vee x_1x_2x_3$.

of f of n variables $x_1, x_2, \ldots, x_n$. Then f is negative with respect to $x_1, \ldots, x_n, e_1, e_2, \ldots, e_t$.

Another property of negativity table is given next.

Theorem 4.4: Each primitive additional column e_j of a negativity table is negative with respect to columns $x_1, x_2, \ldots, x_n$.

Proof: Assume that e_j is added corresponding to the unallowable pair, $f(x^{(j_1)}) = 1$ and $f(x^{(j_2)}) = 0$, to form the negativity table of f. Then $e_j^{(j_1)} = 0$ and $e_j^{(j_2)} = 1$ There is no ℓ such that $x_\ell^{(j_1)} = 0$ and $x_\ell^{(j_2)} = 1$. Hence there must be a subscript r ($1 \leq r \leq n$) such that

$$x_r^{(j_1)} = 1 \text{ and } x_r^{(j_2)} = 0. \qquad (4.8)$$

Condition (2) for a primitive additional column shows that column e_j is negative with respect to columns $x_1, x_2, \ldots, x_n$. Q.E.D.

Theorem 4.3 and Theorem 4.4 indicate that each e_j can be used as a negative function g_j in Fig.4.4. Of course, this realization needs (t+1) negative functions, where t is the number of columns e_j, and may not be minimum.

Definition 4.5: A column c_i is said to cover column c_j if and only if for every entry $c_j^{(k)}$ which is specified,

$$c_i^{(k)} = c_j^{(k)}$$

holds. (If an entry of $c_j^{(k)}$ is not specified, the above equality does not necessarily hold.)

For example, in Table 4.3, c_1 covers c_2 but does not cover c_3. c_3 covers neither c_1 nor c_2, etc.

c_1	c_2	c_3
1		1
0	0	
		0
1	1	1

Table 4.3. Illustration of the covering relation.

The motivation of introducing this concept lies

in the fact that if a column g covers a column e_j of the negativity table and is negative with respect to columns $x_1, \ldots, x_n$, then we can replace e_j by g. (i.e., f is negative with respect to $x_1, \ldots, x_n$, $e_1, \ldots, e_{j-1}$, g, $e_{j+1}, \ldots, e_t$ and g is negative with respect to $x_1, \ldots, x_n$.)

Hence, if g covers more than one e_j, all these e_j can be replaced by g reducing the number of negative functions in the network as shown in Fig.4.4.

Again, the function f given in Tables 4.1 and 4.2 is considered. For this function f, assume the column as shown in Table 4.4.

f	x_1	x_2	x_3	M
1	0	0	0	
1	0	0	1	
1	0	1	0	
0	0	1	1	1
1	1	0	0	
0	1	0	1	1
0	1	1	0	1
1	1	1	1	0

Table 4.4. Realization of $f = \bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_3 \vee x_1x_2x_3$.

As easily checked, M is negative with respect to x_1, x_2, x_3 and moreover M covers e_1, e_2, e_3 shown in

Table 4.2. Therefore, all the three negative functions e_1, e_2, e_3 of variables x_1, x_2, x_3 can be replaced by a single negative function M of variables x_1, x_2, x_3. Fig.4.2 will result by using the negative completion of M (see Table 4.5 in the next section);

$$M^* = \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3.$$

In the next section, a systematic generation of the minimum number of negative functions which together cover all of e_1, e_2, ..., e_t in the negativity table will be discussed.

4.4 Minimum Network by Maximal Compatible Sets

Let us begin with a few new definitions.

Definition 4.6: The conjoint of two columns c_i and c_j, denoted as $c_\ell = c_i \nabla c_j$, is defined by the following rule for each r:

$$c_\ell^{(r)} = c_i^{(r)} \nabla c_j^{(r)},$$

where $c_i^{(r)} \nabla c_j^{(r)}$ is given by

$c_j^{(r)}$ \ $c_i^{(r)}$	1	0	
1	1	?	1
0	?	0	0
	1	0	

The mark ? in the table shows that the operation is not defined. Any blank entry denotes unspecified entry.

If all entries of c_ℓ are defined, (i.e., 1, 0 or blank but not ?) c_ℓ is said definable. The conjoint of more than two columns can be defined by the following inductive formula:

$$c_{i_1} \nabla c_{i_2} \nabla \ldots \nabla c_{i_k} = (c_{i_1} \nabla c_{i_2} \nabla \ldots \nabla c_{i_{k-1}}) \nabla c_{i_k}.$$

Definition 4.7: Two columns c_i and c_j are compatible with respect to $c_{\ell_1}, c_{\ell_2}, \ldots, c_{\ell_s}$ if and only if

$c_i \nabla c_j$ is definable and also negative with respect to $c_{\ell_1}, c_{\ell_2}, \ldots, c_{\ell_s}$.

We will examine the compatibility between each pair of columns e_i and e_j, in the negativity table.

Definition 4.8: Consider a set of columns

$$C_i = \{ c_{i_1}, c_{i_2}, \ldots, c_{i_k} \} . \qquad (4.9)$$

If and only if every two columns in C_i are compatible, with respect to a set of columns B, C_i is called compatible with respect to B. A compatible set C_i covers a column c_j if and only if $c_{i_1} \nabla c_{i_2} \nabla \ldots \nabla c_{i_k}$ covers c_j.

C_i of (4.9) is essentially $c_{i_1} \nabla c_{i_2} \nabla \ldots \nabla c_{i_k}$ according to the following lemma.

Lemma 4.1: C_i defined by (4.9) is compatible with respect to a set of columns B if and only if $c_{i_1} \nabla c_{i_2} \nabla \ldots \nabla c_{i_k}$ is definable and negative with respect to B.

The proof is trivial and hence omitted.

Definition 4.9: Consider all the compatible sets with respect to B. A compatible set C_i is said maximal if and only if among these sets, there is no other compatible set with respect to B, which includes C_i.

Throughout this chapter we will deal with only

compatible sets or maximal compatible sets of primitive additional columns e_i, i = 1, 2, ..., t, in the negativity table with respect to x_1, ..., x_n. Therefore, we will refer to these simply as compatible sets or maximal compatible sets by dropping the phrase "of primitive columns e_i ..... with respect to x_1, ..., x_n".

For example, e_1, e_2 and e_3 of Table 4.2 are pairwise compatible. Hence M = { e_1, e_2, e_3} is a compatible set. Obviously this is maximal.

The concept of maximal compatible sets is important because of the next theorem.

Theorem 4.5: There exists a network with the minimum number of gates for a given function f in which every gate represents a negative function and those in the first level represent negative completions of maximal compatible sets.

Proof: Consider the negativity table of f. e_1, e_2, ..., e_t are all primitive additional columns. Assume that the realization as shown in Fig.4.4 is minimum; i.e., k gates in the first level represent g_1, g_2, ..., g_k and (k+1) is the minimum number of gates which can realize f. Because the last gate represents a negative function with respect to g_1, g_2, ..., g_k, x_1, x_2, ..., x_n, any e_j in the negativity table must be covered by some of g_i. Now let us assume that

g_1 covers e_{11}, ..., e_{1j_1}

g_2 covers e_{21}, ..., e_{2j_2}

g_k covers $e_{k1}, e_{k2}, \ldots, e_{kj_k}$.

Then since each g_i is negative with respect to $x_1, x_2, \ldots, x_n$ and covers $e_{i1} \vee e_{i2} \vee \ldots \vee e_{ij_i}$, $\{ e_{i1}, e_{i2}, \ldots, e_{ij_i} \}$ is a compatible set for i = 1, 2, .., k (Lemma 4.1). Therefore for every $\{ e_{i1}, \ldots, e_{ij_i} \}$, i = 1, 2, ..., k, it is possible to consider a maximal compatible set which includes $\{ e_{i1}, \ldots, e_{ij_i} \}$, respectively. Let us call them as $M_1, M_2, \ldots, M_k$. Finally we can replace each g_i in the network by a negative completion of M_i. After this replacement, f is still negative with respect to $M_1^*, M_2^*, \ldots, M_k^*, x_1, x_2, \ldots, x_n$, where M_i^* is a negative completion of M_i. This network also gives a minimal network since the new network uses the same number of gates as the network of $g_1, g_2, \ldots, g_k$ which is minimal. Q.E.D.

Because of this theorem, the synthesis of a negative function network is done by first generating all the maximal compatible sets and then selecting the minimum number of set from them, which together cover all the primitive additional columns $e_1, e_2, \ldots, e_t$. The generation and selection of maximal compatible sets will be discussed in the next section.

Let us consider the previous example given in Table 4.1. Table 4.2 is the negativity table of f. We have seen that $M = \{ e_1, e_2, e_3 \}$ is the maximal

compatible set and no other maximal compatible set exists. M is the one shown in Table 4.4. Now, we can realize f with two gates, one is for M and the other is the last gate which gives f. A negative completion of M, M*, is shown in Table 4.5.

f	x_1	x_2	x_3	M*
1	0	0	0	1*
1	0	0	1	1*
1	0	1	0	1*
0	0	1	1	1
1	1	0	0	1*
0	1	0	1	1
0	1	1	0	1
1	1	1	1	0

Table 4.5. Completion of M.

The entries of M* with stars were newly specified, following the procedure given in the proof of Theorem 4.2. (All entries were determined uniquely by cases of (1) and (2) of the proof of Theorem 4.2.) M* is written as

$$M^* = \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 .$$

Note that f is now incompletely specified if it is regarded as a function of x_1, x_2, x_3 and M*. A negative completion easily obtained by inspection is

$$f = \bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_3 \vee \bar{M}^* \ .$$

These completions of M and f give the network shown in Fig.4.2. The procedure described so far explained how the minimal network of Fig.4.2 was synthesized.

4.5 Generation and Selection of Maximal Compatible Sets

Let us start this section with a new example whose negativity table is shown in Table 4.6.

f	x_1	x_2	x_3	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
0	0	0	0	1	1	1	1				
1	0	0	1	0							
0	0	1	0					1	1		
0	0	1	1							1	
1	1	0	0		0						
0	1	0	1								1
1	1	1	0			0		0			
1	1	1	1				0		0	0	0

Table 4.6. Negativity table of $f = x_1x_2 \vee x_1\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3$.

From primitive additional columns $e_1, e_2, \ldots, e_t$ specified in a negativity table, we will construct the compatibility matrix of f. The compatibility matrix is a triangular matrix in which columns correspond to $e_1, e_2, \ldots, e_{t-1}$ and rows correspond to $e_2, e_3, \ldots, e_t$, as shown in Table 4.7. A cross is entered if e_i and e_j which correspond to the row and the column of the entry are not compatible. Otherwise it is left as blank. The compatibility matrix of f derived from Table 4.6 is given in Table 4.7.

e_2							
e_3							
e_4							
e_5							
e_6							
e_7	X						
e_8	X	X					
	e_1	e_2	e_3	e_4	e_5	e_6	e_7

Table 4.7. Compatibility matrix of $f = x_1x_2 \vee x_1\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3$.

The compatibility matrix tells us that all pairs of primitive additional columns e_i and e_j are compatible except three pairs (e_1, e_7), (e_1, e_8) and (e_2, e_8).

With the aid of compatibility matrix, we can generate all the maximal compatible sets in the following way. Let us consider primitive additional columns $e_1, e_2, \ldots, e_t$.

(1) Obtain all the maximal compatible sets of e_1 only. Obviously there is only one maximal compatible set $\{e_1\}$.

(2) Given all the maximal compatible sets $M_1, M_2, \ldots, M_b$ of $e_1, e_2, \ldots, e_k$ $(1 \leq k \leq t-1)$, introduce e_{k+1} and form the sets according to the following rule for each i, i = 1, 2, ..., b.

(a) If the set $M_i \cup \{e_{k+1}\}$ is compatible,

i.e., e_{k+1} is compatible with every element of M_i, then form

$$M_i' = M_i \cup \{ e_{k+1} \} ,$$

(b) Otherwise let $\hat{M}_i$ be the set of all the elements of M_i which are compatible with e_{k+1}. Then form two new sets:

$$M_i'' = M_i ,$$
$$M_i''' = \hat{M}_i \cup \{ e_{k+1} \} .$$

(3) Delete all the sets which are included in some other sets generated by (2) above. Note that only M_i''' could be included in other set. The remaining sets are all the maximal compatible sets of $e_1, e_2, \ldots, e_{k+1}$.

(4) Increase k by one. If the new k is equal to t, the procedure is completed, otherwise go to (2) again.

In the above recursive procedure, it is not obvious whether all the maximal compatible sets of $e_1, e_2, \ldots, e_{k+1}$ are generated after completing (3). This will be affirmatively proved in Theorem 4.6. For illustration let us obtain all the maximal compatible sets of $e_1, e_2, \ldots, e_8$ whose compatibility relations are given in Table 4.7. The generation procedure is illustrated in Fig.4.5.

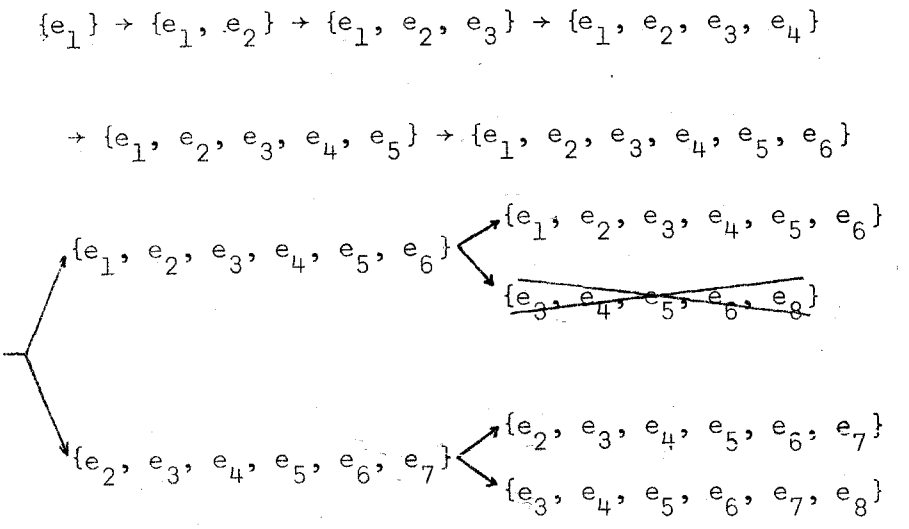


Fig.4.5 Generation of maximal compatible sets of f given by Table 4.6.

Fig.4.5 shows that only one maximal compatible set is generated for each k not more than 6, since any two of $e_1, e_2, \ldots, e_6$ are compatible as seen from Table 4.7. When e_7 is added, two maximal compatible sets result, since procedure (2b) is applied because of non-compatibility of e_1 with e_7. Finally for k = 8, three maximal compatible sets are obtained. $\{ e_3, e_4, e_5, e_6, e_8 \}$ is deleted because this is included in $\{e_3, e_4, e_5, e_6, e_7, e_8 \}$.

Theorem 4.6: Assume that $M_1, M_2, \ldots, M_b$ are all the maximal compatible sets of $e_1, e_2, \ldots, e_k$. All the sets obtained as a consequence of the above steps (2) and (3) are maximal compatible sets of $e_1, \ldots, e_k, e_{k+1}$.

Moreover, there is no other maximal compatible set of $e_1, e_2, \ldots, e_{k+1}$.

Proof: To prove that the above procedure generate all the maximal sets, let us assume that M is a maximal compatible set of $e_1, e_2, \ldots, e_{k+1}$ and is not generated by steps (2) and (3) in the above procedure. M must contain e_{k+1} because otherwise M is equal to one of $M_1, \ldots, M_b$. Let M' be M from which e_{k+1} is deleted. Then M' is included in some of $M_1, \ldots, M_b$ because any two elements in M' are compatible. Let $M_i \supseteq M'$ for some i of i = 1, 2, ..., b. Then M' must be included in or equal to M_i' or M_i''' of step (2). This is a contradiction with the initial assumption that M is maximal and is not generated by steps (2) and (3). Next it is obvious that after step (3) no compatible set which is not maximal remains. Q.E.D.

Definition 4.10: A minimal compatible cover of $e_1, e_2, \ldots, e_t$ for f is a set of the minimum number of maximal compatible sets such that every e_i belongs to at least one of maximal compatible sets chosen.

Now note that to find a minimal compatible cover is equivalent to designing a minimal network of negative functions. However, finding a minimal compatible cover is nothing but a set-covering problem which has been discussed in the literature in conjunction with a variety of problems.

Let $M_1, M_2, \ldots, M_p$ be all the maximal compatible

sets. The covering table is a table, the i-th row of which corresponds to e_i and the j-th column of which corresponds to M_j. The (i, j)-th entry of the covering table is 1 if e_i is covered by M_j, and blank otherwise. The set-covering proplem is to find the minimum number of columns so that every row has an entry 1 in at least one of the selected columns. A number of techniques to solve the set-covering problem through the manipulation of the covering table are discussed in [67], for example.

Another interesting approach is the integer programming approach. The formulation of a set-covering problem by an integer programming is given in [5][29], for example. Recent computational result[48] shows that some problems which correspond* to the cases (30 rows, 50 columns) and (30 rows, 90 columns) are solved in 0.15 seconds and 1.60 seconds respectively on the IBM 360/75I computer.

The covering table for the above example (Table 4.6 and Fig.4.5) is now shown as an example. Let

$$M_1 = \{ e_1, e_2, e_3, e_4, e_5, e_6 \}$$

$$M_2 = \{ e_2, e_3, e_4, e_5, e_6, e_7 \}$$

$$M_3 = \{ e_3, e_4, e_5, e_6, e_7, e_8 \}.$$

---

* The correspondence is not exact in the sense that the matrices are generated randomly and may not represent switching functions.

Then the covering table is:

	M_1	M_2	M_3
e_1	1		
e_2	1	1	
e_3	1	1	1
e_4	1	1	1
e_5	1	1	1
e_6	1	1	1
e_7		1	1
e_8			1

Table 4.8. Covering table of the example given by table 4.6 and Fig.4.5.

Obviously M_1 and M_3 constitute the minimal compatible cover of e_1, e_2, ..., e_8 which is unique in this particular example.

When M_1 and M_3 are considered as functions of variables x_1, x_2, x_3, their negative completions M_1^*, and M_3^* are given in Table 4.9.

M_1^* and M_3^* are written as

$$M_1^* = \bar{x}_1\bar{x}_3$$

$$M_3^* = \bar{x}_1 \vee \bar{x}_2.$$

The minimum network of f using M_1^* and M_3^* as gates in the first level is given in Fig.4.6.

f	x_1	x_2	x_3	M_1^*	M_2^*
0	0	0	0	1	1
1	0	0	1	0	1*
0	0	1	0	1	1
0	0	1	1	0*	1
1	1	0	0	0	1*
0	1	0	1	0*	1
1	1	1	0	0	0
1	1	1	1	0	0

Table 4.9. Completion of M_1 and M_3.

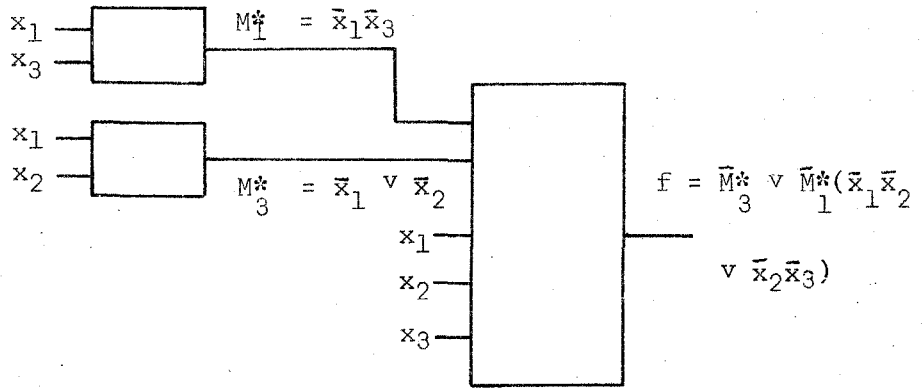


Fig.4.6. A minimum network of f given by Table 4.6.

Up to this point, an entire procedure to obtain a minimum network with a single output has been presented. Next section will deal with a case in which multiple outputs are required.

4.6 Minimal Multiple Output Networks

One of the advatages of our approach is that it can be easily extended to the synthesis of a multiple output network. When we are to realize more than one non-negative function*, only necessary modification is that primitive additional columns e_i are generated in the negativity table for each given function. The rest of procedure, i.e., the generation of maximal compatible sets, the selection of a minimal compatible cover and the completion of each selected maximal compatible set, is exactly the same as that discussed so far. The procedure will also result in a network with the minimum number of gates.

Let us obtain a minimum network of two functions:

$$g = x_1x_2 \vee x_1\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3$$

$$h = x_1\bar{x}_2 \vee x_1\bar{x}_3 \vee \bar{x}_2x_3.$$

The entries given underneath primitive additional columns $e_1, e_2, \ldots, e_9$ in Table 4.10 show from which of g and h the primitive additional column is generated.

---

*When some of functions to be realized are negative, the synthesis would be simpler. In other words each function can be realized by a single gate. Furthermore, the outputs of gates for these negative functions can be used as inputs to other gates which are to realize non-negative functions. The modification of algorithm required is straight-forward, however, and will not be given here.

	g	h	x_1	x_2	x_3	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	M_1^*	M_3^*
(1)	0	0	0	0	0	1	1	1	1	1					1	1
(2)	1	1	0	0	1	0									0	1*
(3)	0	0	0	1	0						1	1			1	1
(4)	0	0	0	1	1								1		0*	1
(5)	1	1	1	0	0		0								0	1*
(6)	0	1	1	0	1			0						1	0	1
(7)	1	1	1	1	0				0		0				0	0
(8)	1	0	1	1	1					0		0	0	0	0	0
						g h	g h	h	g h	g	g h	g	g	g		

Table 4.10. Negativity table of g and h.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
e_2								
e_3								
e_4								
e_5								
e_6								
e_7								
e_8	X							
e_9	X	X	X					

Table 4.11. Compatibility matrix of g and h.

For example, the column e_1 is generated from both g and h. The compatibility matrix is shown in Table 4.11. The generation of all the maximal compatible sets is also shown in Fig.4.7.

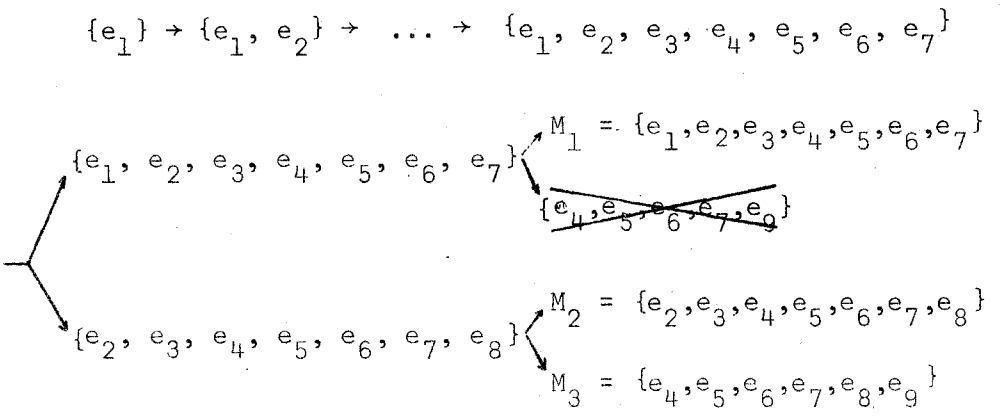


Fig.4.7. Generation of maximal compatible sets of g and h.

Minimal compatible covers can be determined in the same way as the single output network problem. The covering table consists of 9 rows corresponding to $e_1, e_2, \ldots, e_9$ and 3 columns corresponding to M_1, M_2 and M_3. The minimal compatible cover for this problem is $\{M_1, M_3\}$. Negative completions of M_1 and M_3 are also shown in Table 4.10. M_1^* and M_3^* are written as

$$M_1^* = \bar{x}_1\bar{x}_3$$

$$M_3^* = \bar{x}_1 \vee \bar{x}_2.$$

A minimum network of g and h obtained from those M_1^* and M_3^* is given in Fig. 4.8. h does not require M_3^* because all the primitive additional columns e_1, e_2, e_3, e_4, e_6 generated from h can be covered by a single maximal compatible set M_1. (A more rigorous discussion will be found in [50].)

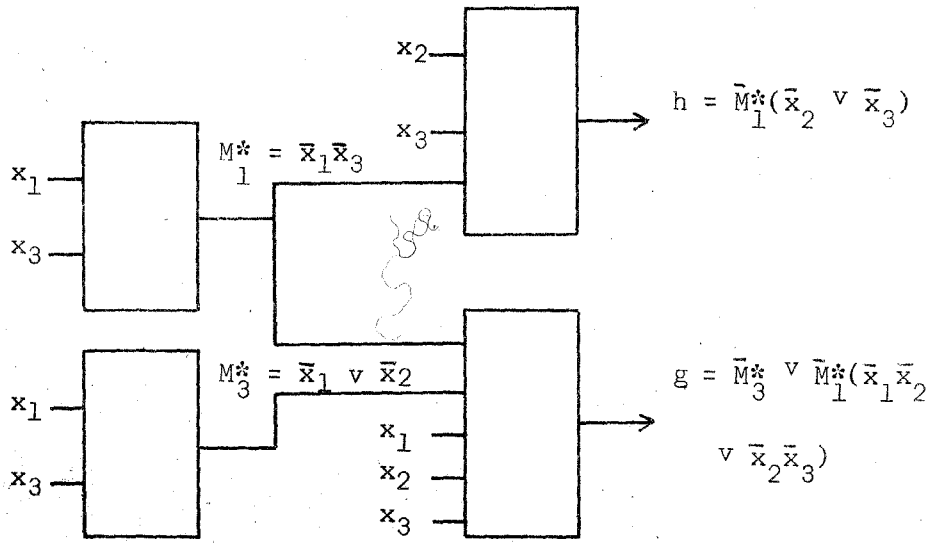


Fig.4.8. Minimum network of g and h.

4.7 Conclusion

The minimization of the number of gates in the network of negative functions was converted into a set-covering problem. Since the set-covering problem is rather easy to solve, this approach appears helpful in obtaining such networks.

It sould be a direction of the future research to develop algorithms based on the other minimality criteria, such as the number of connections, the total cost of gates which is evaluated differently from the number of gates, and also the extension of the approach to more general cases such as feed-forward networks.

Chapter 5 Formulation of Network Synthesis by Integer Programming

5.1 Introduction

Different from the previous two chapters, this chapter discusses about the integer programming formulation of the network design, which is particularly suitable to incorporate network restrictions such as fan-ins restrictions and fan-outs restrictions, and permits a variety of optimality criteria. In the formulation, we assume networks of conventional gates such as NOR, NAND, AND, and OR, or their combinations, but it should be understood that the formulation covers much wider class of gates. Networks of threshold gates would be such examples.

The formulation is then extended to the multiple output networks, by adding extra 0-1 variables. Finally, sequential circuits can also be formulated along this line, including the state assignment.

These problems dealt in this chapter have been recognized as very difficult ones. It is not an exaggeration to state that there has been no systematic approach which is not only theoretically complete but computationally feasible. Of course, it is not claimed that the integer programming approach has provided a final solution, since no integer programming algorithm which works efficiently enough for large scale problems seems available at present. It is hoped, however, that this direction would be a forward step towards the design automation of

logic design, which is greatly needed especially with the advent of integrated circuit and large scale integration. In fact, by means of integer programming approach all the optimal networks of NOR gates, and NOR-AND gates were obtained to verify the practicalness of our approach. Many optimal solutions which represent newly found networks were obtained. The computational aspects of integer programming formulation will be discussed in Chapter 6.

Integer programming problems encountered in this thesis are so-called zero-one all integer programming problems:

$$\begin{aligned} &\text{minimize} && cy \\ &\text{subject to} && A\,y \geq b \qquad (5.1) \\ & && y_j = 0 \text{ or } 1, \\ & && \qquad j = 1, 2, \ldots, N, \end{aligned}$$

where c is an N-dimensional vector, b is an M-dimensional vector and y is an N-dimensional vector of variables. A is an $M \times N$ coefficient matrix.

Any 0 or 1 assignment to all variables is called a solution. Any solutuion such that the constraints of (5.1) are satisfied is called a feasible solution. An optimal solution is a feasible solution such that cy is minimized.

There are several integer programming algorithms available to date. The cutting plane method by Gomory[35][36]

may be applied to this type of problem, though it was originally devised for more general integer programming problems. Recently main effort in this area seems directed to the implicit enumeration algorithm or the branch and bound method, which has been proposed and improved by various authors[4][33][28][29][24][85][48]. We have run both types of algorithms on the computer[6][48]. Since the implicit enumeration algorithm compares much favorably with Gomory's algorithm, judging from the preliminary computational results for our problems, the former is extensively adopted in the computation in Chapter 6.

An integer programming approach was first brought into the design of optimal feed-forward networks of threshold gates by Cameron[11]. His work was followed by Breuer's[9] which considerably simplified the earlier formulation. Muroga also reached essentially the same formulation[77] independently. A more comprehensive aspects were discussed by Muroga and Ibaraki[80].

This thesis, however, does not deal with networks of threshold gates but begin with networks of NOR gates which might be regarded as their special cases. Further possibility is explored in the light of integer programming formulation. Some of the formulations might be merely of theoretical interest for the present, but there is no doubt that future developments within the field of integer programming will gradually eliminate these difficulties.

5.2 Integer Programming Formulation of Networks of NOR Gates.

In this section, a network of NOR gates is described by integer linear inequalities. (NAND gate network can be treated similarly.) The result will be generalized in succeeding sections to deal with other types of networks. Also complete algorithms for deriving optimal networks will be discussed in Section 5.3.

The boolean expression of an NOR gate is

$$\overline{x_1 \vee x_2 \vee \ldots \vee x_n} = \bar{x}_1 \bar{x}_2 \ldots \bar{x}_n .$$

In other words, if at least one of inputs assumes 1 then the output is 0, and 1 otherwise.

In order to describe a network by integer linear inequalities, we have to anticipate a general network which could be reduced to any required network by specifying the actual existence or non-existence of each connection in the anticipated network. For this purpose, we use a feed-forward network. A feed-forward network is a network with R gates which are ordered as gate 1, gate 2, ..., gate R, and each gate can receive inputs from external variables and preceding gates only. The last gate R realizes a given function f. This is illustrated in Fig.5.1. The feed-forward network is the most general network as far as loop-free networks are concerned. Namely, any loop-free network with R gates can result by adequately specifying the existence or non-exiseence of each connection.

Let us assume that f(X) is a function of n variables $x_1, x_2, \ldots, x_n$ with outputs specified for m input vectors $X^{(1)}, \ldots, X^{(m)}$. If f is completely specified, then $m = 2^n$. Also define 0-1 variables as follows. They are illustrated in Fig. 5.2.

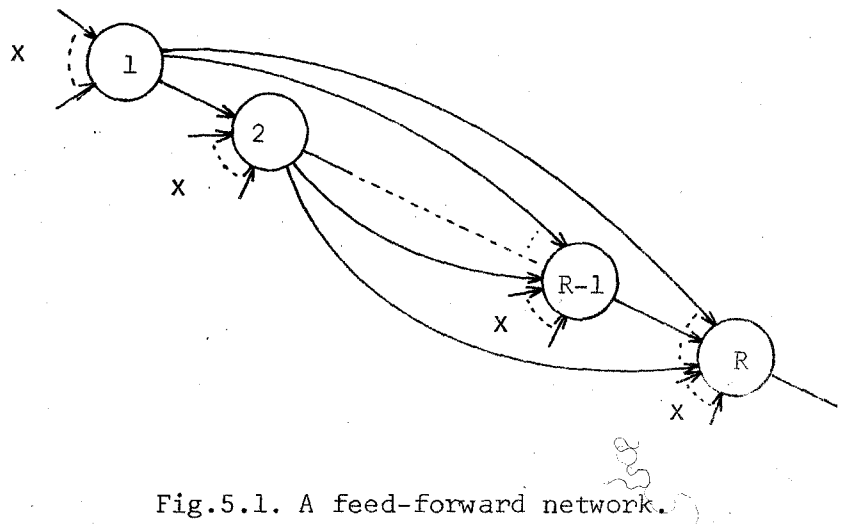


Fig.5.1. A feed-forward network.

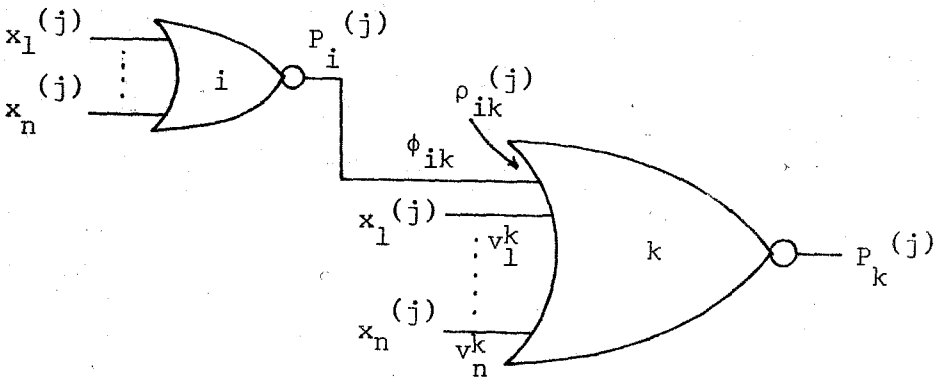


Fig.5.2. Illustration of 0-1 variables in the formulation.

v_ℓ^k : The connection to gate k from x_ℓ. $v_\ell^k = 1$ means the existence of the connection and $v_\ell^k = 0$ means the non-existence.

ϕ_{ik} : The connection from gate i to gate k ($i < k$). $\phi_{ik} = 1$ means the existence of the connection and non-existence otherwise.

$P_k^{(j)}$: The output value of gate k for $x^{(j)}$.

$\rho_{ik}^{(j)}$: The input value of gate k sent from gate i for $x^{(j)}$.

Then the total input value to gate k for $x^{(j)}$ is

$$\sum_{\ell=1}^{n} v_\ell^k x_\ell^{(j)} + \sum_{i=1}^{k-1} \rho_{ik}^{(j)}.$$

From the definition of NOR gates, the output of gate k for $x^{(j)}$ is determined by:

$$\begin{aligned} P_k^{(j)} &= 1 \quad \text{if} \quad \sum_{\ell=1}^{n} v_\ell^k x_\ell^{(j)} + \sum_{i=1}^{k-1} \rho_{ik}^{(j)} \leq 0 \\ P_k^{(j)} &= 0 \quad \text{if} \quad \sum_{\ell=1}^{n} v_\ell^k x_\ell^{(j)} + \sum_{i=1}^{k-1} \rho_{ik}^{(j)} \geq 1. \end{aligned} \tag{5.2}$$

This relation can be converted to linear inequalities of 0-1 variables;

$$\begin{aligned} -\sum_{\ell=1}^{n} v_\ell^k x_\ell^{(j)} - \sum_{i=1}^{k-1} \rho_{ik}^{(j)} &\geq -U(1 - P_k^{(j)}) \\ \sum_{\ell=1}^{n} v_\ell^k x_\ell^{(j)} + \sum_{i=1}^{k-1} \rho_{ik}^{(j)} &\geq 1 - UP_k^{(j)}, \end{aligned} \tag{5.3}$$

$$j = 1, 2, \ldots, m$$
$$k = 1, 2, \ldots, R-1, \qquad (5.3)^*$$

where U is a sufficiently large positive number such that if $P_k^{(j)} = 0$, then the upper inequality of (5.3) is non-binding for any assignment of 0-1 variables, and if $P_k^{(j)} = 1$, then the lower inequality has the similar property. For example,

$$U = n + k - 1$$

suffices this requirement. Hereafter, the letter U is used solely for this purpose. Although the actual amount of value U required for each inequality may vary, we do not distinguish individually but use the same letter U.

From the definition of U, it may be obvious that (5.2) and (5.3) are equivalent, i.e., feasible solutions of (5.3) are exactly those which satisfy relation (5.2). Note that (5.3) is linear in all variables v_ℓ^k, $\rho_{ik}^{(j)}$, $P_k^{(j)}$, since $x_\ell^{(j)}$ are given constants.

Now, let us consider the variable $\rho_{ik}^{(j)}$. This is related to the output of gate i by

$$\rho_{ik}^{(j)} = \phi_{ik} P_i^{(j)}. \qquad (5.4)$$

In other words, $\rho_{ik}^{(j)} = 1$ if and only if $P_i^{(j)} = 1$ and gates i and k are connected. This non-linear form can

---

* $\sum_{A}^{B}$, $A < B$, is defined as 0.

also be converted into a linear form as follows:

$$
\begin{aligned}
& -P_i^{(j)} - \phi_{ik} + \rho_{ik}^{(j)} \geq -1 \\
& P_i^{(j)} + \phi_{ik} - 2\rho_{ik}^{(j)} \geq 0, \qquad (5.5) \\
& \qquad k = 2, 3, \ldots, R \\
& \qquad i = 1, 2, \ldots, k-1 \\
& \qquad j = 1, 2, \ldots, m.
\end{aligned}
$$

The proof that (5.4) and (5.5) are equivalent is straightforward if we take into account that every variable assumes only 0 or 1.

The last gate is treated saparately since its bahavior is specified by the output values of function f. The total input value of the last gate for $x^{(j)}$ has to satisfy the following inequalities:

$$
\begin{aligned}
& -\sum_{\ell=1}^{n} v_\ell^k x_\ell^{(j)} - \sum_{i=1}^{k-1} \rho_{ik}^{(j)} \geq 0 \quad \text{for } f(x^{(j)}) = 1 \\
& \qquad\qquad (5.6) \\
& \sum_{\ell=1}^{n} v_\ell^k x_\ell^{(j)} + \sum_{i=1}^{k-1} \rho_{ik}^{(j)} \geq 1 \quad \text{for } f(x^{(j)}) = 0 \ , \\
& \qquad j = 1, 2, \ldots, m.
\end{aligned}
$$

All inequalities (5.3) (5.5) and (5.6) characterize the entire feed-forward network. Namely, any feasible solution of the system of inequalities (5.3), (5.5) and (5.6) provides a loop-free NOR network with R gates which realizes the given function f. In case f requires more than R gates, the system of inequalities (5.3), (5.5)

and (5.6) is infeasible, i.e., it has no feasible solution. However, since NOR gates constitute a complete set of functions[86][51], there is a number of gates R_f for any f, which is sufficient to realize the function f.

Further aspects of the formulation and whole algorithm to obtain optimal networks under a certain criterion will be discussed in Section 5.3 and 5.4.

5.3 Incorporation of Network Restrictions

Some of the restrictions imposed on a network can be taken into account by simply adding inequalities. Ease of incorporation of restriction is one of the characteristics of the integer programming approach.

The restrictions on the maximum number of inputs (fan-ins) for each gate can be considered as follows. Let I be the maximum fan-ins of gate k. Then, obviously

$$\sum_{\ell=1}^{n} v_{\ell}^{k} + \sum_{i=1}^{k-1} \phi_{ik} \leq I \qquad (5.7)$$

limits the number of input connections of gate k to at most I.

The maximum number of outputs from each gate, the maximum fan-outs, can also be treated similarly by;

$$\sum_{i=k+1}^{R} \phi_{ki} \leq J. \qquad (5.8)$$

Next, let us consider the restrictions on the number of levels of a network. Let us assume that networks with more than K levels are prohibited. Then there arise sets of connections which are not permissible because of the level restriction. For example, the set of connections,

$$\phi_{12} = 1, \ \phi_{23} = 1, \ \ldots, \ \phi_{KK+1} = 1$$

is not permitted bacause K consecutive connections indicate the existence of more than K levels. This can be

eliminated by adding the inequality:

$$\phi_{12} + \phi_{23} + \ldots + \phi_{K\,K+1} \leq K-1. \qquad (5.9)$$

This inequality assures at least one of ϕ_{12}, ϕ_{23},..., $\phi_{K\,K+1}$ is 0, thus destroying the consecutive K connections.

From this argument, a procedure to keep a network within K levels is :

(1) Exhaust all sets of connections which give more than K levels.

(2) For every such set, add an inequality to prohibit the set of connections, derived in a similar way as (5.9).

Other types of restrictions on conncetions can usually be incorporated. For example, if a connection $\phi_{ik} = 1$ is prohibited for some reason, simply add

$$\phi_{ik} = 0.$$

The conditional restriction, i.e., the restriction such that $\phi_{i'k'} = 0$ if $\phi_{ik} = 1$ is realized by:

$$\phi_{i'k'} \leq 1 - \phi_{ik}.$$

The argument used for the level restricted network could be extended to more involved conditions such as the planarity of a network. A procedure is first to list all conditions which lead to the non-planarity, and then to set up corresponding inequalities to prevent them.

If R and n increases, however, the number of inequalities will become excessively large, making the formulation impractical. One approach to avoid this difficulty is:

(1) Solve the set of inequalities without adding the extra inequalities which guarantees, say, the planarity, if incorporated.

(2) If a solution obtained shows a planar network, then terminate. Otherwise go to (3).

(3) Pick up some extra inequalities among those prohibiting the planarity and add to the system of inequalities used in (2) to make the current solution infeasible. Then solve the new problem resulted.

By repeating (3) we will eventually obtain an planar network.

5.4 Procedures to Design Optimal Networks

We have so far discussed the integer linear constraints which together characterize a feed-forward NOR network, and also those which impose network restrictions. Any feasible solution for the resulting system of inequalities represents a NOR network for a given function f subject to the imposed network restrictions. This section discusses procedures to obtain optimal networks among those feasible solutions, under certain optimality criteria.

First consider, as an example, a combined optimality criterion such that the primal objective is the minimization of the number of gates, and the secondary objective is the minimization of the number of connections.

Note that the number of connections for a feed-forward network with R gates is written as:

$$\sum_{k=1}^{R}\left(\sum_{\ell=1}^{n} v_{\ell}^{k} + \sum_{i=1}^{k-1} \varphi_{ik}\right) \qquad (5.10)$$

Procedure 1:

(1) Set R = 1. (If we know a lower bound of the number of gates required for realizing a given function f, set R to this number.)

(2) Formulate a set of inequalities for an R gate feed-forward network for a given function f, and possible inequalities corresponding to the restrictions imposed on the network. Solve this integer program with an objective function (5.10). If this has an optimal

solution, it gives an optimal network under the above combined optimality criterion. Otherwise go to (3).

(3) Increase R by 1 and return to (2).

The whole procedure will terminate in a finite number of steps, if the problem is feasible for some R and an integer programming algorithm with finite convergence is employed for solving each resulting problem.

It may be obvious that the above procedure yields an optimum network, because the first R which satisfies (2) is the minimum number of NOR gates which is sufficient for realizing f.

An alternative procedure for deriving an optimal feed-forward network under the same optimality criterion is outlined in the following.

First introduce new 0-1 variables λ_k such that

$$\sum_{i=k+1}^{R} \phi_{ki} \leq U\lambda_k, \qquad (5.11)$$

$$k = 1, 2, \ldots, R-1.$$

If at least one of ϕ_{ki}, $i = k+1, \ldots, R$ is 1, then λ_k assumes 1, otherwise (5.11) is non-binding. As a result, if we minimize

$$\Lambda = \sum_{k=1}^{R-1} \lambda_k, \qquad (5.12)$$

then Λ shows the number of gates actually used, since λ_k assumes 0 if the left hand side of (5.11) is 0. In this case, $\Lambda + 1$ is the total number of gates actually used, since (5.12) excludes the last gate which must be always used to provide the required function f.

Procedure 2:

First prepare a sufficiently large number of gates which will guarantee the realizability of a given function by the network. Then set up a set of inequalities which characterizes the network and also incorporate the network restriction. Solve the integer program with the minimization of objective function*

$$\sum_{k=1}^{R-1} \lambda_k . \qquad (5.13)$$

An optimum solution of this integer programming problem provides a network with the minimum number of gates. Alternatively, if we use the following objective function,

$$V \sum_{k=1}^{R-1} \lambda_k + \sum_{k=1}^{R} \left(\sum_{\ell=1}^{n} v_\ell^k + \sum_{i=1}^{k-1} \phi_{ik} \right) \qquad (5.14)$$

instead of (5.13), then an optimal solution in the sense that the networks are those with the minimum number of connections among networks with the minimum number

* $\sum_{k=1}^{R-1} e_k \lambda_k$ may be used, if gate k has cost e_k, and the total cost is the design objective.

gates will be obtained. V in (5.14) is a sufficiently large positive number such that

$$V \geq nR + R(R-1)/2.$$

Although this procedure needs only one integer program, the size of the problem is relatively larger than that of Procedure 1. Judging from the fact that the convergence of most of integer programming algorithms become progressively slow with the increase of the size of problems, Procedure 1 may be preferred at present.

An advantage of Procedure 2 is that the other types of objective functions can be used as the primary objective. For example, the use of

$$\sum_{k=1}^{R} \left(\sum_{\ell=1}^{n} v_{\ell}^{k} + \sum_{i=1}^{k-1} \phi_{ik} \right) \qquad (5.15)$$

as the objective function yields the networks with the minimum number of connections regardless of the number of gates.

In Section 5.2, the number of levels was treated from the view point of network restriction. Sometimes, however, it should be considered as a design objective to obtain a fast circuit. For this purpose, a feed-next network is desirable as a basis model rather than a feed-forward network. A feed-next network is a network with E levels, in which each gate in the j-th level, j = 1, 2, ..., E,can receive inputs only from

external variables and gates in the preceding levels. An example is illustrated in Fig.5.3. The number of gates

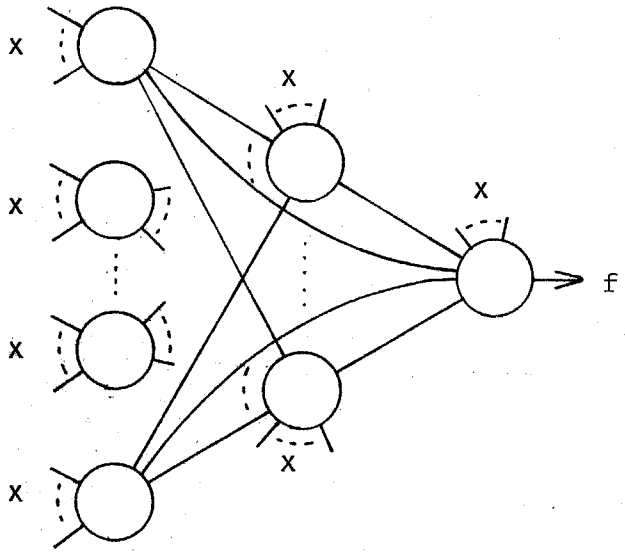


Fig.5.3 Feed-next network.

in each level sufficient for a given function is sometimes available from the theoretical consideration. Otherwise, however, a tentative number would be used in the formulation. A variant of Procedure 1 for this case is as follows:

Procedure 1':

(1) Set E = 1.

(2) Formulate a set of inequalities for a E level feed-next network. Solve the resulting integer program with a certain objective function such as the number of connections. If this has

an optimal solution, it gives an optimal network. Otherwise, go to (3).

(3) Increase E by one, and return to (2).

Procedure 2 for a feed-forward network can also be modified to deal with a feed-next network, though it is omitted here.

In concluding this section, it should be mentioned that any other types of gates can be considered as constituents of the network if they are represented in inequalities. For example, NAND networks can be treated similarly. The case in which mixture of different gates is permitted, such as mixture of NOR gates and AND gates, will be discussed in the subsequent sections.

5.5 Feed-Forward Networks Using Mixture of Different Gates

As an extension of previous sections, we will consider a feed-forward network in which a mixture of different types of gates is permitted. A feed-forward network in which each gate is either NOR gate or AND gate is such an example. Let us assume that B different types of gates realizing functions $g_1, g_2, \cdots, g_B$ are permitted in a network. Each gate is called a module. Namely, each module in a feed-forward network assumes one of $g_1, g_2, \cdots, g_B$.

This type of problem is of practical importance provided that network restictions are taken into consideration, and that a variety of optimality criteria can be used. There has been no systematic approaches for this problem except for certain special cases.

In order to formulate this problem as an integer program, we first introduce a conceptual module, called a generative module. A set of parameters is associated with each generative module. Depending upon the values of parameters, the generative module represents one of $g_1, g_2, \cdots, g_B$. For illustrative purpose (Fig.5.4), it is convenient to consider that a generative module consists of two parts: one is a multi-output network with outputs $g_1, g_2, \cdots, g_B$, and the other is a selection network which selects one of $g_1, g_2, \cdots, g_B$ according to the parameters associated with it.

An outline of our procedure is : (1) to formulate

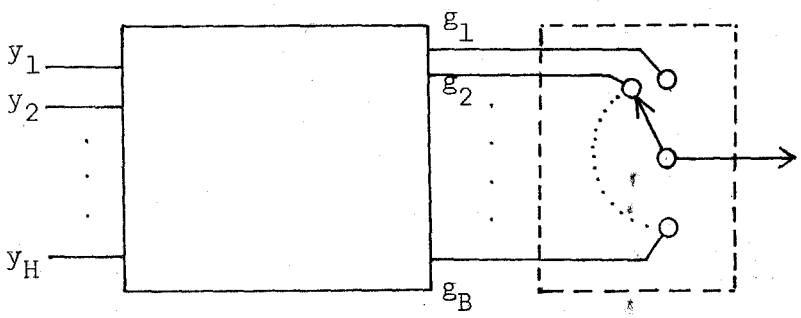


Fig.5.4. A generative module.

a feed-forward network with R generative modules in integer programming. Solutions for this problem will determine the connection between generative modules and parameters for each module as well. (2) If we obtain a feasible solution for a given function f with the minimum R, then replace each generative module by the actual gate which the generative module implies.

In Section 5.6, an integer programming formulation of a generative module will be derived. Then in Section 5.7, an entire integer programming formulation of a feed-forward network will be presented.

5.6 Integer Inequalities for a Generative Module

The first step to derive a generative module is to express all given $g_1, g_2, \ldots, g_B$ by integer linear inequalities. The selection of one module out of these in each generative module will be discussed later.

One method will be given below to express g_1 through g_B by a single network of threshold gates. Other methods based on a multi-threshold threshold gate and based on the direct specification of true and false vectors of each of $g_1, \ldots, g_B$ by integer linear inequalities will be given in [81].

Let us introduce a number H such that

$$H = \max (p_1, p_2, \ldots, p_B),$$

where p_i is the number of input terminals of the module corresponding to g_i. By adding idle input terminals, we can assume that every module has H input terminals. H represents the maximum fan-ins restriction on each module.

Now, given functions g_1 through g_B of H variables, let us realize these functions by a multiple-output network of threshold gates as shown in Fig.5.5. Our aim is to derive the relation between H input variables $y_1, y_2, \ldots, y_H$ and B output functions $g_1, g_2, \ldots, g_B$, in the form of integer linear inequalities. Although any network which realizes $g_1, g_2, \ldots, g_B$ may be used to express a generative module, a network with the minimum number of threshold gates is preferable because it

gives a concise expression. Threshold gates are used simply as a mathematical tool, no matter what physical devices are employed to implement $g_1, g_2, \cdots, g_B$. The threshold gate network may be obtained by using the integer linear programming approach[80]. Of course any other method known so far can be used for this purpose. Several methods will be found in [10][76]. The method discussed in Chapter 3 may be used.

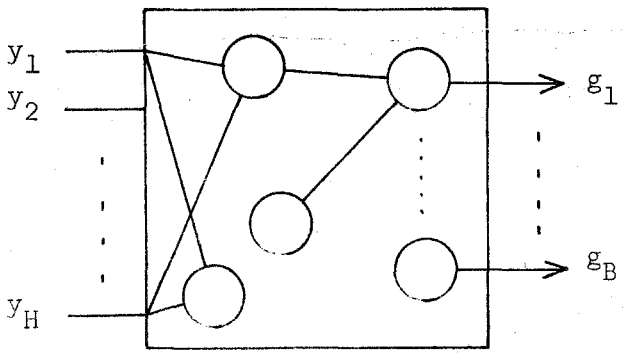


Fig.5.5. Collective expression of $g_1, g_2, \cdots, g_B$ by a multiple output network of threshold gates.

Let us number threshold gates from 1 to C in a synthesized feed-forward network with multiple outputs in which each gate receives inputs only from the preceding gates and variables $y_1, y_2, \cdots, y_H$. Assume that the C_r-th gate realizes g_r, $r = 1, 2, \ldots, B$.

Let the output value of the k-th gate for the input vector $y^{(j)} = (y_1^{(j)}, \ldots, y_H^{(j)})$ be $Q_k^{(j)}$, $j = 1, \ldots, m$. Let α_{ik} denote the weight from the i-th gate to the k-th gate, w_ℓ^k denote the weight of the variable y_ℓ to the k-th gate and T_k denote the threshold of the k-th gate. Note that α_{ik}, w_ℓ^k and T_k are known constants. But $y_\ell^{(j)}$, and $Q_i^{(j)}$ are unknown variables.

With these notations defined, the k-th gate in the network for each k is expressed in inequalities;

$$\sum_{\ell=1}^{H} w_\ell^k y_\ell^{(j)} + \sum_{i=1}^{k-1} \alpha_{ik} Q_i^{(j)} \geq T_k - U\,(1-Q_k^{(j)})$$

$$-\sum_{\ell=1}^{H} w_\ell^k y_\ell^{(j)} - \sum_{i=1}^{k-1} \alpha_{ik} Q_i^{(j)} \geq -T_k + 1 - U\,Q_k^{(j)}, \qquad (5.16)$$

$$j = 1, 2, \ldots, m.$$

These inequalities are interpreted in the similar manner as those of NOR networks. Namely, (5.16) are converted into the following;

$$\Sigma\, w_\ell^k y_\ell^{(j)} + \Sigma\, \alpha_{ik} Q_i^{(j)} \geq T_k \quad \text{if} \quad Q_k^{(j)} = 1$$

$$\Sigma\, w_\ell^k y_\ell^{(j)} + \Sigma\, \alpha_{ik} Q_i^{(j)} \leq T_k - 1 \quad \text{if} \quad Q_k^{(j)} = 0.$$

When the network is synthesized such that the C_r-th gate realizes g_r, $Q_{C_r}^{(j)}$ represents $g_r(y^{(j)})$. The set of inequalities (5.16) for k = 1, 2, .., C, together with the relation

$$Q_{C_r}^{(j)} = g_r(y^{(j)}),$$

$$j = 1, 2, \ldots, m,$$

$$r = 1, 2, \ldots, B,$$

represent the functional relations between $y_1, \ldots, y_H$ and $g_1, \ldots, g_B$. Thus we have formulated the aggregate of $g_1, \ldots, g_B$ in integer linear inequalities. This is the first step of our derivation of a generative module.

Example 5.1: Let us find an integer linear inequality expression of two modules:

$$g_1 = y_1 \oplus y_2$$

$$g_2 = y_1 y_2.$$

These two functions g_1 and g_2 can be realized by two threshold gates connected as shown in Fig. 5.6.

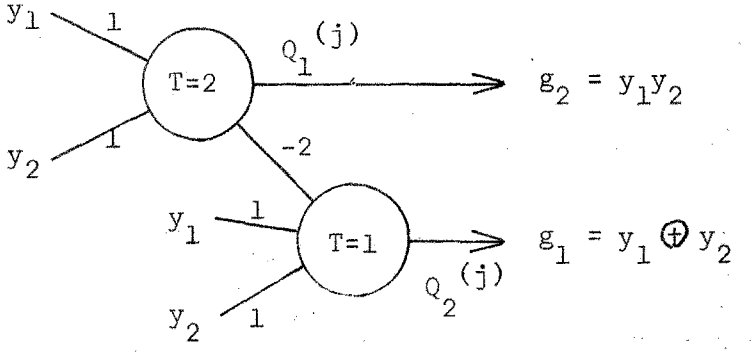


Fig.5.6. Realization of two modules given in Example 5.1.

According to the above discussion, the outputs of threshold gates $Q_1^{(j)}$ and $Q_2^{(j)}$ for each $y^{(j)}$, $j = 1, 2, \ldots, m$, can be defined by the following inequalities:

$$y_1^{(j)} + y_2^{(j)} \geq 2 - U(1 - Q_1^{(j)})$$

$$- y_1^{(j)} - y_2^{(j)} \geq -1 - UQ_1^{(j)},$$

$$j = 1, 2, \ldots, m.$$

$$y_1^{(j)} + y_2^{(j)} - 2Q_1^{(j)} \geq 1 - U(1 - Q_2^{(j)})$$

$$- y_1^{(j)} - y_2^{(j)} + 2Q_1^{(j)} \geq 0 - UQ_2^{(j)},$$

$$j = 1, 2, \ldots, m.$$

Any value of $U \geq 3$ is sufficient for these inequalities. If we consider that

$$g_1(y^{(j)}) = Q_2^{(j)}$$

$$g_2(y^{(j)}) = Q_1^{(j)} \qquad (5.17)$$

for each $y^{(j)}$, $j = 1, 2, \ldots, m$, these inequalities describe relations between the input values and the output values of g_1 and g_2.

Now let us consider the selection of one function out of $g_1, \ldots, g_B$. As selection parameters, introduce variables $\theta_1, \ldots, \theta_B$ whose values are 1 or 0 such that if $\theta_r = 1$, then a generative module represents the

corresponding module g_r. In order to select only one module out of $g_1, \ldots, g_B$, the following equality is added.

$$\theta_1 + \theta_2 + \ldots + \theta_B = 1. \qquad (5.18)$$

Let $P^{(j)}$ denote the value which the output of the generative module assumes for the j-th input vector $y^{(j)}$. In other words, $P^{(j)}$ satisfies the relation

$$P^{(j)} = g_r(y^{(j)}) = Q_{C_r}^{(j)}, \qquad (5.19)$$

if $\theta_r = 1$ holds, where $Q_{C_r}^{(j)}$ is the output value of the threshold gate which realizes g_r. This is expressed in inequalities;

$$P^{(j)} \leq Q_{C_r}^{(j)} + (1 - \theta_r)$$

$$P^{(j)} \geq Q_{C_r}^{(j)} - (1 - \theta_r), \qquad (5.20)$$

$$r = 1, 2, \ldots, B$$

$$j = 1, 2, \ldots, m.$$

Obviously, if $\theta_r = 0$ then the inequalities become non-restrictive and if $\theta_r = 1$ the inequalities force $P^{(j)}$ to be equal to $Q_{C_r}^{(j)}$.

Consequently, with inequalities (5.16), (5.18) and (5.20) together, a generative module which is able to choose one of $g_1, \ldots, g_B$ is completely described. This expression will be the basis for the optimal network synthesis procedure in the following sections.

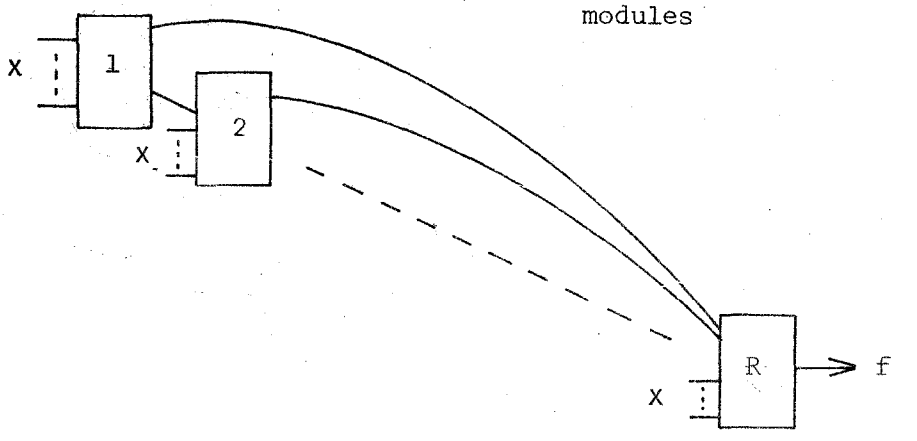
5.7. Description of a Feed-Forward Network Consisting of Generative Modules

In this section we will describe a feed-forward network of generative modules in linear inequalities of integral variables.

A. General Case

Consider a feed-forward network consisting of R generative modules, as shown in Fig. 5.7.

Fig. 5.7. Feed-forward network of R generative modules



Assume that the function of a generative building block is not symmetric in variables. Symmetric case will be discussed in B. The input terminals of each generative module must be numbered for identification.

Let $\phi_i(d, k)$ denote a (0, 1)-variable which expresses the connections from the output of the d-th generative module to the i-th input terminal of the k-th module. If $\phi_i(d, k) = 0$, it means no connection and if $\phi_i(d, k) = 1$, the connection exists. However note that there may be

more than one connection from the d-th module to the k-th module*. In other words, we may have $\phi_{i_1}(d, k) = \phi_{i_2}(d, k)$ $= \cdots = 1$ for the same d and k. Let $v_i(\ell, k)$ denote a (0, 1)-variable which expresses the connection of an external variable x_ℓ to the i-th input terminal of the k-th generative module. Here

$$k = 1, 2, \ldots, R$$

$$d = 1, 2, \ldots, k-1$$

$$i = 1, 2, \ldots, H$$

$$\ell = 1, 2, \ldots, n, +, -.$$

$v_i(+, k)$ is the connection from the constant input of the value 1 and $v_i(-, k)$, the connection from the constant input of the value 0.

Each input terminal receives exactly one connection, i.e., for each k and each i

$$\sum_{d=1}^{k-1} \varphi_i(d,k) + \sum_{\ell=1}^{n,+,-} v_i(\ell,k) = 1. \qquad (5.21)$$

The input value $y_i^{(j)}(k)$ at the i-th input terminal of the k-th generative module for the j-th external input vector $x^{(j)}$ satisfies the following condition,

---

* A degenerate function such as $x_1 \vee x_2$ by substituting x_1 into x_3 of $x_1x_3 \vee x_2$ would be generated.

$$y_i^{(j)}(k) = \begin{cases} P^{(j)}(d) & \text{if } \varphi_i(d, k) = 1 \\ x_\ell^{(j)} & \text{if } v_i(\ell, k) = 1, \end{cases} \qquad (5.22)$$

$$(i = 1, 2, \ldots, H)$$

$$(k = 1, 2, \ldots, R).$$

where $P^{(j)}(d)$ is the output of the d-th generative module for the j-th external input vector $x^{(j)}$ and

$$x_+^{(j)} = 1 \text{ and } x_-^{(j)} = 0 . \qquad (5.23)$$

(5.22) is expressed in the following set of linear inequalities of integral variables:

$$y_i^{(j)}(k) \leq P^{(j)}(d) + (1 - \varphi_i(d,k))$$

$$P^{(j)}(d) \leq y_i^{(j)}(k) + (1 - \varphi_i(d,k)), \qquad (5.24)$$

$$(k = 1, 2, \ldots, R)$$

$$(d = 1, 2, \ldots, k-1)$$

$$(i = 1, 2, \ldots, H)$$

$$(j = 1, 2, \ldots, m),$$

and

$$y_i^{(j)}(k) \leq x_\ell^{(j)} + (1 - v_i(\ell, k))$$

$$x_\ell^{(j)} \leq y_i^{(j)}(k) + (1 - v_i(\ell, k)), \qquad (5.25)$$

$$(k = 1, 2, \ldots, R)$$

$$(\ell = 1, 2, \ldots, n, +, -)$$

$$(i = 1, 2, \ldots, H)$$

$$(j = 1, 2, \ldots, m).$$

$\phi_i(d, k) = 0$ makes (5.24) non-restrictive and $\phi_i(d, k) = 1$ yields $y_i^{(j)}(k) = P^{(j)}(d)$, and so forth.

Note that the external variables $x_\ell^{(j)}$ are given in advance as input vectors to the whole network and considered as constants in our integer linear programming approach. The outputs of the R-th module are also specified in advance as the values of the given function $f(x^{(j)})$ for all j's. Therefore $P^{(j)}(R)$ are also given constants but $P^{(j)}(k)$ for $k \neq R$ are unknown variables.

B. Networks with Symmetric Generative Modules

If each generative module is symmetric in all inputs, in other words, if the input terminals of each module are interchangeable, then each input terminal need not be differentiated from others. Let us sum up the input values at the k-th module and denote it with

$$y^{(j)}(k) = y_1^{(j)}(k) + \cdots + y_H^{(j)}(k). \qquad (5.26)$$

In the following, only $y^{(j)}(k)$ will be used, eliminating the use of $y_i^{(j)}(k)$'s. The value which $y^{(j)}(k)$ assumes is not limited to 0 and 1. Also the subscript i may be dropped from $\phi_i(d, k)$ and $v_i(\ell, k)$. $\phi(d, k)$ and $v(\ell, k)$ are variable which represent the connection from the d-th module to the k-th module and the connection from the ℓ-th external variable to the k-th module, respectively. However note that in the current case $\phi(d, k)$ and $v(\ell, k)$ also are not restricted to the value, 1 or 0, because multiple connections, in other words, the simultaneous connections from an output of a module to more than one input terminal of a certain module may yield a better network. However the total number of inputs to each module must equal H, i.e.

$$\sum_{d=1}^{k-1} \varphi(d,k) + \sum_{\ell=1}^{n,+,-} v(\ell,k) = H. \qquad (5.27)$$

$y^{(j)}(k)$ is expressed as follows. Let us introduce new variables $\rho^{(j)}(d, k)$ and $\rho^{(j)}(\ell, k)$ which satisfy

$$\left.\begin{aligned} \rho^{(j)}(d,\ k) &\leq \varphi(d,\ k) \quad + U\ (1 - P^{(j)}(d)) \\ \varphi(d,\ k) &\leq \rho^{(j)}(d,k) + U\ (1 - P^{(j)}(d)) \\ \rho^{(j)}(d,\ k) &\leq U\ P^{(j)}(d), \end{aligned}\right\} \quad (5.28)$$

$$(k = 1, 2, \ldots, R)$$

$$(d = 1, 2, \ldots, k-1)$$

$$(j = 1, 2, \ldots, m),$$

$$\left.\begin{aligned} \rho^{(j)}(\ell,\ k) &\leq v\ (\ell,\ k) + U\ (1 - x_\ell^{(j)}) \\ v(\ell,\ k) &\leq \rho^{(j)}(\ell,k) + U\ (1 - x_\ell^{(j)}) \\ \rho^{(j)}(\ell,\ k) &\leq U\ x_\ell^{(j)}, \end{aligned}\right\} \quad (5.29)$$

$$(\ell = 1, 2, \ldots, n, +, -)$$

$$(j = 1, 2, \ldots, m)$$

$$(k = 1, 2, \ldots, R).$$

These inequalities of (5.28) and (5.29) imply

$$\rho^{(j)}(d,\ k) = \varphi\ (d,\ k)\ P^{(j)}(d) \quad (5.30)$$

$$\rho^{(j)}(\ell,\ k) = v\ (\ell,\ k)\ x_\ell^{(j)}. \quad (5.31)$$

Since the input of the k-th module is the sum of contributions from the outputs of other modules and the external

variables, $y^{(j)}(k)$ of (5.26) which consists of $y_i(j)$ of (5.22) is obtained as

$$y^{(j)}(k) = \sum_{d=1}^{k-1} \varphi(d, k)\, P^{(j)}(d) + \sum_{\ell=1}^{n,+,-} v(\ell, k)\, x_{\ell}^{(j)}$$

$$= \sum_{d=1}^{k-1} \rho^{(j)}(d, k) + \sum_{\ell=1}^{n,+,-} \rho^{(j)}(\ell, k) \qquad (5.32)$$

using (5.30) and (5.31). Consequently the connections in a network are described by inequalities, (5.27), (5.28), (5.29), (5.32) together.

5.8. Procedures to Design an Optimal Feed-Forward Network

We are now ready to discuss design procedures of an optimal feed-forward network, with all the gates and the network described in inequalities. The behavior of generative modules which can express any of the given modules $g_1, \ldots, g_B$ was expressed in inequalities in Section 5.6, and the connections from other generative modules and the external variables in Section 5.7. The set of all these inequalities together with constants $x^{(j)}$ and $f(x^{(j)})$, i.e., $P^{(j)}(R)$ specifies the condition which the feed-forward network must satisfy. Any feasible solution of this set of inequalities realizes the given function f. However if a certain objective function is introduced and optimized (minimized or maximized), then an integer programming problem is formulated. If a design objective such as the number of connections can be expressed as a linear function and chosen as an objective function, then an optimum network with the minimum number of connections will be obtained.

Assume that the feed-forward network for f has R generative modules. Only one of g_r's is chosen in each generative module in an optimum solution of our integer linear program. Let J and K be the numbers of the 0-1 variables and the inequalities, respectively, which are necessary in obtaining the relations between the inputs $y_1^{(j)}(k), \ldots, y_H^{(j)}(k)$ and the outputs $P^{(j)}(k)$ of the k-th generative module. (For example, (5.16) (5.18) and (5.20)) The set of these inequalities necessary for each generative module will be referred to as E. Then the

numbers of the variables in the entire formulation for this network are:

m H R	for	$y_i^{(j)}(k)$'s
m(R-1)	for	$P^{(j)}(k)$'s
$\frac{HR(R-1)}{2}$	for	$\varphi_i(d, k)$'s
H R (n+2)	for	$v_i(\ell, k)$'s

and the total number of the 0-1 variables is

$$RJ + mHR + m\ (R-1) + \frac{HR(R-1)}{2} + HR(n+2). \qquad (5.33)$$

Note that the number of $P^{(j)}(k)$ is m(R-1) because

$$P^{(j)}(R) = f\ (\ x^{(j)}) \qquad (5.34)$$

and therefore $P^{(j)}(R)$ are not counted. The number of the inequalities are

2HR	for	(5.21)*
mHR(R-1)	for	(5.24)
2mHR(n+2)	for	(5.25).

---

* An equality is expressed in two inequalities

The total number of the inequalities is

$$RK + 2\,HR + mHR(R-1) + 2\,mHR(n+2). \tag{5.35}$$

As discussed in Section 5.7B, if the modules are symmetric in input variables, the numbers of variables and inequalities can be fewer than those given above.

To obtain a feed-forward network with R modules which realizes a given function f, we need solve the inequalities E for every module and also (5.21), (5.24) and (5.25) under the condition $P^{(j)}(R) = f(x^{(j)})$.

$$\sum_{k=1}^{R} \sum_{i=1}^{H} \left(\sum_{d=1}^{k-1} \varphi_i(d, k) + \sum_{\ell=1}^{n} v_i(\ell, k) \right) \tag{5.36}$$

may be chosen as an objective function. This implies the minimization of the number of connections between modules and from external variables excluding those of the constants in the network. Other objectives may be chosen instead, as an objective function.

A procedure for obtaining a feed-forward network with the minimum number of modules and then, as the secondary objective, the minimum number of connections (i.e. a network with the minimum number of connections among those with the minimum number of modules) is as follows:

Procedure I

(1) Set R = 1. (or set R to a lower bound of value necessary to realize f if it can be found by certain means.)

(2) Solve the set E of the inequalities, (5.21), (5.24) and (5.25), for the network of R generative modules, minimizing the objective function (5.36). If there is a solution, an optimum solution of this problem gives an optimal network for f. If there is no solution, go to (3).

(3) Increase R by one and return to (2).

This procedure will terminate in a finite number of steps, if the given set of modules, g_1, ..., g_B, are complete*.

Obviously, this is a generalization of Procedure I for NOR networks described in Section 5.4. Procedure II in the same section can also be generalized in the similar way. The latter approach would have merits such that it permit a wider variety of objective functions than Procedure I. However, the detailed discussion will not be given here. In [81], its approach and other aspects of the integer programming formulation is discussed in some detail.

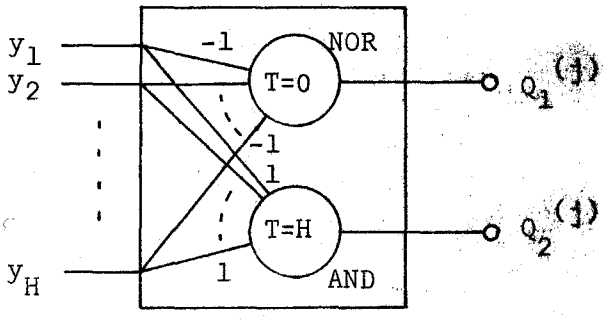
---

* If any function can be realized by combining g_1,...., g_B, the set is called complete. See [51].

5.9. Example - NOR-AND Networks -

As an example of network consisting of more than one type of module, we consider a network with NOR gates and AND gates.

Fig. 5.8 Expression of NOR and AND.



First of all, let us consider the linear inequality expression for a generative module which can express either a NOR gate or an AND gate (Section 5.6). In this formulation, we do not impose the fan-in restriction* on each module.

Now consider an expression for NOR gate in the k-th generative module of the network. Obviously, the relation

---

* The k-th generative module is assumed to have exactly H (=(k-1)+n) input terminals for each k. If a module turns out to have actually fewer input terminal connections, connect constants of appropriate values to the remaining input terminals.

between inputs $y_r^{(j)}$, r = 1, 2, ..., H (= (k-1)+ n), and its output $Q_1^{(j)}$ is written as follows, as illustrated in Fig. 5.8:

$$-\sum_{r=1}^{H} y_r^{(j)} \geq 0 - U\,(1 - Q_1^{(j)}) \tag{5.37}$$

$$\sum_{r=1}^{H} y_r^{(j)} \geq 1 - U\,Q_1^{(j)}, \qquad j = 1, 2, \ldots, m.$$

Taking advantage of the property that a NOR gate is symmetric in all variables, we can group all the inputs into two sets; one is to receive the external variables and the other is to receive the outputs from the preceding modules (Section 5.7B). (5.37) is then rewritten as

$$-\sum_{\ell=1}^{n} v(\ell,k)\, x_\ell^{(j)} - \sum_{d=1}^{k-1} \varphi\,(d,k)\, P^{(j)}(d) \geq 0 - U\,(1-Q_1^{(j)})$$

$$\sum_{\ell=1}^{n} v(\ell,k)\, x_\ell^{(j)} + \sum_{d=1}^{k-1} \varphi\,(d,k)\, P^{(j)}(d) \geq 1 - U\,Q_1^{(j)}, \tag{5.38}$$

$$(j = 1, 2, \ldots, m),$$

where $\Sigma\, v(\ell,k)\, x_\ell^{(j)} + \Sigma\, \phi\,(d,k)\, P^{(j)}(d)$ denotes the sum of contributions from the external variables and the outputs of the other modules. However, these inequalities are not linear because $\phi(d,k)$ and $P^{(j)}(d)$ are both unknown

variables. Therefore new variables $\rho^{(j)}(d,k)$, such that

$$\rho^{(j)}(d,k) = \varphi\,(d,k)\; P^{(j)}(d) \tag{5.39}$$

are introduced (see (5.30) and (5.32)) and (5.38) becomes the new set of linear inequalities:

$$\begin{aligned} &-\sum_{\ell=1}^{n} v\,(\ell,k)\; x_{\ell}^{(j)} - \sum_{d=1}^{k-1} \rho^{(j)}(d,k) \;\geq\; 0 - U(1 - Q_1^{(j)}) \\ &\sum_{\ell=1}^{n} v\,(\ell,k)\; x_{\ell}^{(j)} + \sum_{d=1}^{k-1} \rho^{(j)}(d,k) \;\geq\; 1 - U\; Q_1^{(j)}, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (5.40) \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j = 1, 2, \ldots, m. \end{aligned}$$

$v(\ell,k)\; x_{\ell}^{(j)}$ is not substituted by $\rho^{(j)}(\ell,k)$ as in (5.31) because this is already linear ($x_{\ell}^{(j)}$'s are given constants). To express the relation (5.39) in linear form, the following is used instead of (5.28) in order to reduce the number of inequalities.

$$\begin{aligned} &-P^{(j)}(d) - \varphi\,(d,k) + \rho^{(j)}(d,k) + 1 \geq 0 \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (5.41) \\ &P^{(j)}(d) + \varphi\,(d,k) - 2\,\rho^{(j)}(d,k) \geq 0 \end{aligned}$$

This is possible because $\rho^{(j)}(d,k)$ is known to be 0 or 1 in this case.

In a similar way, an AND gate of the k-th generative module is written as follows:

$$\sum_{\ell=1}^{n} v(\ell,k)\ x_\ell^{(j)} + \sum_{d=1}^{k-1} \rho^{(j)}(d,k)$$

$$\geq \sum_{\ell=1}^{n} v(\ell,k) + \sum_{d=1}^{k-1} \phi(d,k) - U\,(1 - Q_2^{(j)})$$

$$- \sum_{\ell=1}^{n} v(\ell,k)\ x_\ell^{(j)} - \sum_{d=1}^{k-1} \rho^{(j)}(d,k)$$

$$\geq - \sum_{\ell=1}^{n} v(\ell,k) - \sum_{d=1}^{k-1} \phi(d,k) + 1 - U\,Q_2^{(j)}, \qquad (5.42)$$

$$j = 1, 2, \ldots, m\ ,$$

where $Q_2^{(j)}$ is the output value of the AND gate.

Combining (5.40) and (5.42), and introducing a selection variable $\theta(k)$ (see Section 5.6), the k-th generative module is described by

$$- \sum_{\ell=1}^{n} v(\ell,k)\ x_\ell^{(j)} - \sum_{d=1}^{k-1} \rho^{(j)}(d,k)$$

$$\geq 0 - U\,(1 - P^{(j)}(k)) - U\,(1 - \theta(k)),$$

$$\sum_{\ell=1}^{n} v(\ell,k)\, x_{\ell}^{(j)} + \sum_{d=1}^{k-1} \rho^{(j)}(d,k)$$

$$\geq 1 - U\, P^{(j)}(k) - U\,(1 - \theta(k)),$$

$$\sum_{\ell=1}^{n} v(\ell,k)\, x_{\ell}^{(j)} + \sum_{d=1}^{k-1} \rho^{(j)}(d,k)$$

$$\geq \sum_{\ell=1}^{n} v(\ell,k) + \sum_{d=1}^{k-1} \phi(d,k) - U\,(1 - P^{(j)}(k)) - U\,\theta(k),$$

$$- \sum_{\ell=1}^{n} v(\ell,k)\, x_{\ell}^{(j)} - \sum_{d=1}^{k-1} \rho^{(j)}(d,k)$$

$$\geq -\sum_{\ell=1}^{n} v(\ell,k) - \sum_{d=1}^{k-1} \phi(d,k) + 1 - U\, P^{(j)}(k) - U\,\theta(k), \tag{5.43}$$

$$(k = 1, 2, \ldots, R-1)$$
$$(j = 1, 2, \ldots, m),$$

where $P^{(j)}(k)$ is the output value of the k-th generative module for the j-th external input vector. $\theta = 1$ means that NOR is selected and $\theta = 0$ means that AND is selected. For the last generative module,

$$- \sum_{\ell=1}^{n} v(\ell,R)\, x_{\ell}^{(j)} - \sum_{d=1}^{R-1} \rho^{(j)}(d,R) \geq 0 - U\,(1 - \theta(R)), \tag{5.44}$$

$$\sum_{\ell=1}^{n} v(\ell,R)\, x_{\ell}^{(j)} + \sum_{d=1}^{R-1} \rho^{(j)}(d,R)$$

$$\geq \sum_{\ell=1}^{n} v(\ell,R) + \sum_{d=1}^{R-1} \phi(d,R) - U\,\theta(R) \qquad \text{for } f(x^{(j)}) = 1$$

or

$$\sum_{\ell=1}^{n} v(\ell,R)\, x_\ell^{(j)} + \sum_{d=1}^{R-1} \rho^{(j)}(d,R) \geq 1 - U\,(1 - \theta(R)), \tag{5.45}$$

$$- \sum_{\ell=1}^{n} v(\ell,R)\, x_\ell^{(j)} - \sum_{d=1}^{R-1} \rho^{(j)}(d,R)$$

$$\geq - \sum_{\ell=1}^{n} v(\ell,R) - \sum_{d=1}^{R-1} \phi(d,R) + 1 - U\,\theta\,(R),$$

$$\text{for } f(x^{(j)}) = 0,$$

$$(j = 1, 2, \ldots, m).$$

Also, to describe the non-linear relation $\rho^{(j)}(d,k) = P^{(j)}(d)\ \phi\ (d,k)$ in linear form, (5.41) is necessary for

$$k = 1, 2, \ldots, R$$

$$d = 1, 2, \ldots, k-1$$

$$j = 1, 2, \ldots, m.$$

Consequently (5.41), (5.43), (5.44) and (5.45) together completely represent a network consisting of NOR and AND gates.

Note that the formulation, described in the previous sections, was simplified in sevelal respects by considering particular properties of NOR-AND network.

For example, selection variables $\theta_1(k)$ and $\theta_2(k)$ respectively for NOR and AND, are reduced to a single variable $\theta\ (k)$ by using

$$\theta_1(k) + \theta_2(k) = 1, \tag{5.46}$$

of (5.18). Variables $Q_1^{(j)}$ and $Q_2^{(j)}$ for the k-th generative module are also replaced by a single variable $P^{(j)}(k)$ for the following reason. If $\theta(k) = 1$, say (i.e. NOR is represented by a generative module), only inequalities for a NOR gate (the first two of (5.43)) becomes restrictive and others for an AND gate becomes non-restrictive. Therefore, even if the restriction $Q_1^{(j)} = Q_2^{(j)} = P^{(j)}(k$ is incorporated, it does not change the nature of the inequalities.

As explained in the paragraph after (5.40), $\rho^{(j)}(\ell,k)$ for external variables are also deleted. A significant reduction of the number of variables $y^{(j)}(k)$'s is obtained by substituting $y^{(j)}(k)$ by (5.32).

In most cases, if types of modules are concretely given, we can reduce the size of problem, by considering special properties of the types. However, we will not try a detailed discussion about these possibilities here.

With all these inequalities, all the optimum networks for each three variable switching function which have the minimum number of modules and, secondary, the minimum number of connections were obtained. Procedure 1 in Section 5.8 was used for computation. The computational aspects will be discussed in Section 6 together with the design of NOR networks.

5.10. Multiple Output Networks

One of the important but more difficult problem is the design of a multiple output network in which more than one function is simultaneously realized. In this section, two approaches are presented. One is an extension of the feed-forward network formulation and the other is all-interconnection network formulation which is quite efficiently solved when combined with the implicit enumeration algorithm of integer programming.

A. Feed-Forward Network Formulation

Assume that S functions $f_1, f_2, \ldots, f_S$ are to be realized by a feed-forward network with R gates (modules). Let the output value of the k-th module ($1 \leq k \leq R$) for the j-th input vector be $P_k^{(j)}$. Then the design of a multiple output network requires that for any f_a, $1 \leq a \leq S$, there exists a moude b such that

$$P_b^{(j)} = f_a(x^{(j)}), \qquad (5.47)$$

$$1 \leq b \leq R,$$

$$j = 1, 2, \ldots, m.$$

The characterization of $P_k^{(j)}$ has been discussed in previous sections under several assumptions. For example, $P_k^{(j)}$ in a feed-forward NOR network can be described by (5.3) and (5.5). However, a slight modification is required for the last gate R, because we do not know which function of $f_1, f_2, \ldots, f_S$ the last module realizes. Therefore, 0-1 variables $P_R^{(j)}$ is also introduced and

(5.3) should be set up for k = 1, 2, ..., R - 1, R.

In case of networks with the mixture of different gates, $P_k^{(j)}$ (this is written as $P^{(j)}(k)$), k = 1, 2, ..., R, is given by (5.21), (5.24), (5.25) and E for every module, without the condition

$$P^{(j)}(R) = f(x^{(j)})$$

which was necessary for the single output network.

Now let us consider the relation (5.47) and express it in inequalities. Introduce a set of new 0-1 variables

$$\psi_{kt} \qquad (5.48)$$
$$k = 1, 2, \ldots, R$$
$$t = 1, 2, \ldots, S,$$

where $\psi_{kt} = 1$ if the output of the k-th module realizes the function f_t, and $\psi_{kt} = 0$ otherwise. Then

$$\sum_{k=1}^{R} \psi_{kt} = 1, \qquad (5.49)$$
$$t = 1, 2, \ldots, S,$$

holds because each f_t is realized be exactly one module in the network.

Now if $\psi_{kt} = 1$, then

$$P_k^{(j)} = f_t(x^{(j)})$$

must follow. This is written in the following inequalities:

$$f_t(x^{(j)}) \leq P_k^{(j)} + U\,(1 - \psi_{kt})$$
$$P_k^{(j)} \leq f_t(x^{(j)}) + U\,(1 - \psi_{kt}) \qquad (5.50)$$
$$t = 1, 2, \ldots, S$$
$$k = 1, 2, \ldots, R$$
$$j = 1, 2, \ldots, m_t,$$

where m_t is the number of specified input vectors for f_t.

Consequently, given a feed-forward network, the inequalities which characterize R modules in the network, (5.49) and (5.50) are all the necessary inequalities to describe a multiple output network.

A procedure which results in optimal multiple output networks with the primal objective that the minimization of the number of gates, and with a certain secondary objective such as the number of connections is as follows:

(1) Set R = S (or any other lower bound of the number of gates in a network, if available).

(2) Set up inequalities which characterize a feed-forward network with R gates and S functions. Solve this integer programming problem with the objective function representing the secondary objective mentioned above. If this problem has feasible solutions, optimal solutions give optimal multiple output networks. Otherwise go to (3).

(3) Increase R by 1. Then return to (2).

Of course, other procedures discussed in Section 5.4 in conjunction with NOR networks can be extended to this case. Other primal objectives can be optimized provided that an appropriate procedure is employed.

B. All-Interconnection Network Formulation

Different from the feed-forward network used as a basis for the design procedure, this approach assumes an all-interconnection network. An all-interconnection network is a network with R modules in which the connection from any gate to any gate in the network is allowed. It

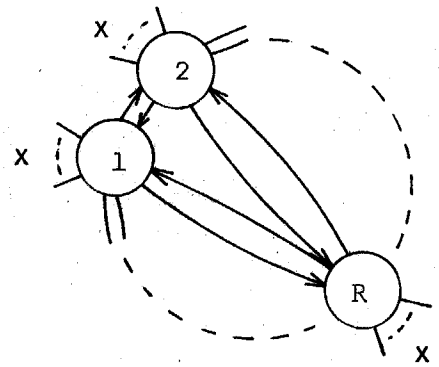


Fig.5.9. All-interconnection network.

is illustrated in Fig. 5.9.

An advantage of designing a multiple output network by means of all-interconnection network formulation is that we can assign each function f_t, t = 1, 2, ..., S, (S ≤ R), to module t, i.e.,

$$P_t^{(j)} = f_t(x^{(j)}) \qquad t = 1, 2, \ldots, S. \tag{5.51}$$

This is possible because each module in the all-interconnection network is symmetrically related with the rest. Thus

selection parameter ψ_{kt} used in A can be deleted in this formulation.

However, the employment of all-interconnection networks does not mean that the resulting network for $f_1, f_2, \ldots, f_S$ is permitted to contain connection loops. Although networks with loops may realize the functions in much simpler form, it usually needs satisfy other involved conditions to guarantee stable operations. Thus, in this thesis, the loops are simply excluded from our condideration.

Therefore, the restrictions to supress the occurence of loops must be added. This can also be done by inequalities. As an illustration, assume a loop consisting of $\phi_{12} = 1$, $\phi_{23} = 1$, $\phi_{31} = 1$. This is eliminated by adding an inequality

$$\phi_{12} + \phi_{23} + \phi_{31} \leq 2. \qquad (5.52)$$

This example suggests a method for supressing loops in an all-interconnection network: (1) list up all the possible loops in an all-interconnection network, and (2) set up an inequality derived in the same way as (5.52) for every possible loop.

The number of such inequalities may grow excessively large as R and n increase. Another method is possible, which is not based on the inequalities, when the implicit enumeration algorithm is used for the integer programming problem, though it is not described here.

The set of inequalities which characterizes an all-

interconnection network can be derived in a similar way as that of a feed-forward network. As an example, an all-interconnection network with R NOR gates is considered here. Let define 0-1 variables v_ℓ^k, ϕ_{ik}, $P_k^{(j)}$, $\rho_{ik}^{(j)}$ as given in Seftion 5.2. Considering that gate k can receive input from any other gates, the behavior of gate k is described by:

$$-\sum_{\ell=1}^{n} v_\ell^k x_\ell^{(j)} - \sum_{i\neq k} \rho_{ik}^{(j)} \geq -U\,(1 - P_k^{(j)}) \tag{5.53}$$

$$\sum_{\ell=1}^{n} v_\ell^k x_\ell^{(j)} + \sum_{i\neq k} \rho_{ik}^{(j)} \geq 1 - U\,P_k^{(j)},$$

$$k = 1, 2, \ldots, R$$

$$j = 1, 2, \ldots, m,$$

where m is the total number of input vectors which are specified for at least one function of $f_1, f_2, \ldots, f_s$. To realize the relation

$$\rho_{ik}^{(j)} = \phi_{ik}\, P_i^{(j)}$$

in linear form, the following inequalities are needed.

$$-P_i^{(j)} - \phi_{ik} + \rho_{ik}^{(j)} \geq -1 \tag{5.54}$$

$$P_i^{(j)} + \phi_{ik} - 2\rho_{ik}^{(j)} \geq 0,$$

$$k = 1, 2, \ldots, R$$

$$i \neq k$$

$$j = 1, 2, \ldots, m.$$

These inequalities (5.53) and (5.54), as well as those suppressing loops characterize an all-interconnection network with R gates. If this set of inequalities under condition (5.51) has feasible solutions, they correspond to loop-free networks which realize a given set of functions $f_1, f_2, \ldots, f_S$.

The procedures to obtain optimal networks can be derived by considering in the similar way as in other cases. For example, the procedure obtained by replacing the word "feed-forward network" by "all-inteconnection network" in Procedure of A of this section is a procedure to obtain optimal networks with primal objective of minimizing the number of gates, and with a certain secondary objective.

The all-interconnection network formulation presented here is valid not only for multiple output networks but also for single output networks. When combined with the implicit enumeration algorithm of integer programming, the all-inerconnection network formulation is favorably compared with the feed-forward network formulation in computation speed. The formulation will be explored again from this viewpoint in Chapter 6.

5.11 Design of an Optimal Sequential Network

Synthesis of sequential network with the minimization of a certain design objective such as the number of gates is known to be very difficult mainly bacause of the addition of state assignment problem. Here we will show that this problem also can be formulated as an integer linear programming problem.

When we design a sequential network, various restrictions such as the realization by a shift register or the realization by certain types of flip-flops are usually imposed on a network. As most of these restricrions can be incorporated by adding extra linear inequalities, only general formulation will be discussed.

We consider a problem of designing a sequential network with the minimum number of states. In other words, first minimize the number of states in the state transition table according to the conventional switching theory (the state minimization technique which most of textbooks[67][32] of switching theory discuss) and then synthesize an optimum network with the most appropriate state assignment in the sense that it requires the fewest number of gates under the minimum state assumption. Since the optimization of the network and the state assignment will be made by integer programming approach only after the number of the states is minimized, however, the resulting networks may not have the minimum number of gates if it is considered as the primal objective.

On the other hand, integer programming makes it possible

to synthesize an optimal sequential network using the input-output relation rather than the state transition table. In this case, an appropriate state transition table and state assignment such that the number of gates in a network is minimized will be made simultaneously by integer programming approach. However, it is not described in this thesis. See [81].

As shown in Fig.5.10, our sequential network is assumed to consist of a combinational network and loops with delays. The combinational network is a multiple

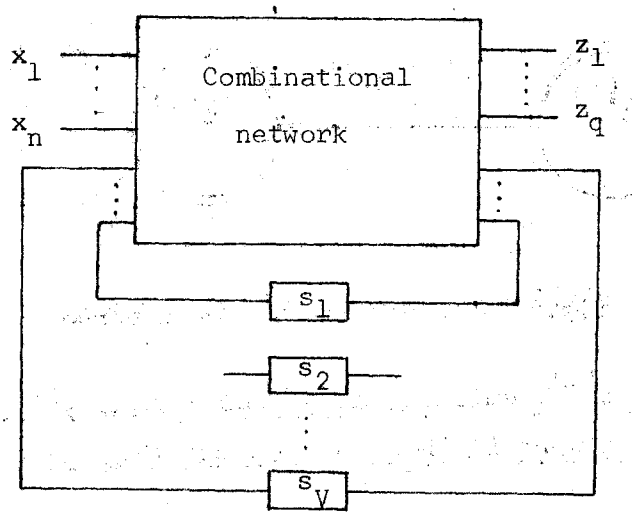


Fig.5.10. Realization of a sequential network.

output feed-forward network which has been discussed in the previous sections.

Each s_i is an internal variable which is fed back to the inputs of the network with a unit time delay.

The vector $s = (s_1, s_2, \ldots, s_v)$ is called the (internal) state of this sequential network. $x_1, x_2, \ldots, x_n$ are (external) input variables and $z_1, z_2, \ldots, z_q$ are (external) output variables. s_i, x_j, z_k assume the value 0 or 1.

As known in switching theory, the state transition diagram completely specifies the behavior of the sequential network. The transition shown in Fig.5.11 shows that if the present state is $s^{(i)}$ and the external input vector $x^{(j)}$ is applied to the network, then the network will be in

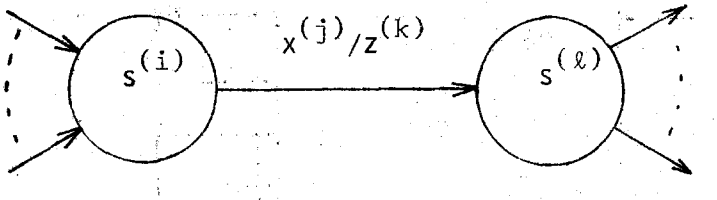


Fig.5.11. State transition diagram

the next state $s^{(\ell)}$, emitting the output vector $z^{(k)}$. The combinational network in Fig.5.10 realizes this relation, i.e.,

$$z^{(k)} = f(s^{(i)}, x^{(j)})$$

$$s^{(\ell)} = h(s^{(i)}, x^{(j)}), \qquad (5.55)$$

where f and h are vectors* whose coordinates are functions of $s^{(i)}$ and $x^{(j)}$. Introducing f and h, we have essentially a multiple-output network with V+n inputs (the external and internal inputs togerher) and V + q outputs. The difference of the network from that discussed in the previous sections is that $x^{(j)}$ and $z^{(k)}$ are specified in advance but $s^{(i)}$ are unknown variables.

An integer linear programming problem for disigning a multiple-output network whose outputs $s^{(i)}$ are fed back to the inputs of the network through the relation (5.55) can be directly formulated with technique developed in the previous sections.

First the number of the states in a state transition diagram which is either completely or incompletely specified is minimized. Let the minimum number of the states be D, and find Γ such that

$$\Gamma = \langle \log_2 D \rangle \qquad (5.56)$$

where $\langle x \rangle$ is the integer colosest but not smaller than x. Then set V to Γ. In other words, consider the sequential network of Fig.5.10 with V = Γ internal variables.

Assume that the transition shown in Fig.5.11 takes place, in other words, when the state of the network is in $s^{(i)}$ and the inputs are $x^{(j)}$, the next state is $s^{(\ell)}$ and the outputs become $z^{(k)}$. Now consider the multiple

---

*f has q coordinates each of which is a function of (V+n) varialbes. h has V coordinates each of which is a function of (V+n) variables.

output feed-forward combinational network with the $n+\Gamma$ inputs, $x^{(j)}$ and $s^{(i)}$, and the $q + \Gamma$ outputs, $z^{(k)}$ and $s^{(\ell)}$, breaking all the feed-back lines of Fig.5.10. Then derive the inequalities for this multiple output network according to the procedure of the previous sections for each transition in the state transition diagram. In these inequalities $s^{(i)}$ and $s^{(\ell)}$ are kept unknown variables while $x^{(j)}$ and $z^{(k)}$ are known constants.

The fact that vairables $s^{(i)}$ are unknown causes no difficulty because the formulation discussed so far is already linear in the input variables $x_{\ell}^{(j)}$ and the output variables $P^{(j)}(k)$. (See (5.24) and (5.25))

When the state transition diagram is incompletely specified, some coordinates of $s^{(i)}$ and $z^{(k)}$ are not specified. Accordingly the corresponding equalities for (5.55) are eliminated. Therefore the resulting multiple output functions, f and h, are generally incompletely specified.

Let us describe a procedure for designing a sequential network which has the minimum nember of modules (these modules constitute the combinational network in Fig.5.10) under the conditions that the network has the minimum number of states and that the network has the minimum number of feed-back lines.

Let R be the number of generative modules in the network. The procedure is as follows:

(1) Set $R = \Gamma$ (or set R to a lower bound of the number of modules which is found by some means).

(2) Solve the integer linear programming problem for a multiple output feed-forward combinational network*with R modules which has $(n + \Gamma)$ inputs and $(\Gamma + q)$ outputs, optimizing an objective function. If it has feasible solutions, we have found optimal networks. If not go to (3).

(3) Increase R by 1 and return to (2).

If the number of connections is chosen as the objective function, we can derive a sequential network with the minimum number of connections among the networks of the minimum number of modules.

---

*One gate in the network may correspond more than one of $(\Gamma + q)$ output functions.

5.12 Conclusion

Some important problems in logical design, which are known to be very difficult, were formulated by integer programming approach. These problems include the design of NOR networks, networks with other type of gates or with their combinations. Also sequential networks fall in this category.

An advantage of integer programming approach is the capability of handling a wide variety of optimality criteria, such as the number of gates, the number of connections and the number of levels in the network. The easy incorporation of network restrictions such as fan-ins and fan-outs restrictions would be another advantage.

Since the size of the integer program grows very rapidly as R and n increases, the computationally feasible problems are limited. However, several important logical design problems were actually tried with certain success. In the next chapter, the oomputational results for NOR networks and NOR-AND networks will be reported as well as several improvements attained in the course of the computation experiment.

It may be said that these problems are computationally feasible. A number of interesting optimum networks were obtained for the first time by integer programming approach.

This thesis is intended to show a main idea utilized in the integer programming formulation and its versatility.

Some detailed discussions which may be important in practice are not included in the thesis. More detailed discussions and a few further applications will be found in [80][81].

Chapter 6 Computational Aspects of Network Designs by Integer Programming

6.1 Introduction

Based on the integer programming approach discussed in Chapter 5, optimal combinational networks of NOR gates and those of NOR-AND gates have been synthesized. This chapter presents computational results as well as a number of gimmicks used for improving the computation speed. Since Gomory's algorithm does not work satisfactorily, judging from a preliminary computational result [6], the integer programming algorithm used for solving those problems is the implicit enumeration method. The algorithm actually used is described in detail elsewhere[48]. It is further tailored to our problems by making use of their particular structures, improving the computational efficiency greatly over the first algorithm.

All the optimal networks of three variable switching functions with NOR gates and those with the combination of NOR-AND gates are exhausted by integer programming approach. The initial computation time on IBM 360/75I with the H level FORTRAN IV compiler for NOR networks for all three variable switching function (80 representative functions of equivalent classes by permutation of variables.) is 110 minutes, and it is further improved by the factor of about 10 times by using the above second algorithm. It takes 54 minutes for synthesizing optimal NOR-AND combination network for all three variable functions.

These results may suggest the computational feasibility of integer programming approach and encourages further investigation.

6.2 Integer Programming and Implicit Enumeration

This section presents an outline of the integer programming problem and implicit enumeration algorithm for solving it. For detailed description, see references [4][28][48] for example. The computer code used for the network synthesis is discussed in reference [63]. It is a result of simplification and modification of the original implicit enumeration algorithm[4][33][24][85] in order to improve the computational efficiency.

An integer programming problem with N unknown variables and M constraints is in general stated as follows again:

$$\text{minimize} \quad c\,y$$

$$\text{subject to} \quad A\,y \geq b \qquad (6.1)$$

where c is an N-dimensional vector of non-negative constants, b is an M-dimensional vector of constants and A is an M x N matrix of given coefficients, and y is an n-dimensional vector of variables. In our case, all variables y are integers which assume only 1 or 0. Sometimes, this is refered to as a zero-one integer programming problem.

The implicit enummeration algorithm has been computationally proved to be one of the most efficient methods for solving this type of zero-one problem. It implicitly enumerates all the 2^N solutions without explicitly and exhaustively examining all of them, and picks up the best feasible solution.

Let us start with several definitions. When all the

variables in y are assigned 1 or 0 it will be called a solution. If a solution satisfies the constraints of (6.1), it will be called a feasible solution and if not, an infeasible solution. A feasible solution that minimizes cy is an optimum feasible solution. A partial solution S is defined as an assignment of binary values to a subset of the N variables. Any variables which are not assigned are called free variables. A completion of a partial solution S is a binary assignment to all free variables.

Let us outline the implicit enumeration algorithm as it is shown in Fig. 6.1. With a given partial solution S^0 and the incumbent solution (the feasible solution having the smallest value of the objective function obtained thus far), the block entitled "CHK-IEQ" is entered. At this point, examine whether some of the free variables must be 1 or 0 if each inequality is to be satisfied. Scanning through the inequalities until no more free variables are assigned, S^0 with these free variables assigned becomes a new partial solution S^1. Next the partial solution S^1 is checked to determine which of the following three cases occurs.

(1) Feasible: The completion of S^1 obtained by setting all free variables to 0 is found to be feasible. It is compared with the incumbent and the better of the two is maintained as the incumbent. The backtrack procedure is initiated to obtain a new partial solution S^2 by changing some of the assigned variables

according to a certain rule.*

(2) Infeasible: If at least one inequaliy is not satisfied by S^1 whatever binary values are assigned to the free variables, then S^1 is discarded immediately by initiating the backtrack procedure. The backtrack procedure forms a new partial solutions S^2 from S^1.

(3) Augment S^1: If neither of the above two cases occurs, a free variable is assigned to 1, forming S^2. The choice of this variable greatly affects the convergence and it should be made according to the type of problem being solved.

After replacing S^2 with S^0, the entire procedure is repeated by reentering the block "CHK-IEQ".

By cycling through this procedure repeatedly, the computation results in the implicit enumeration of all possible solutions. When the computation terminates, the incumbent is an optimal solution. The checking procedure of each inequality such that one of cases (1), (2) and (3) is quickly identified is explained in [48]. The implicit enumeration algorithm converges in a finite number of steps, but the efficiency of the algorithm heavily depends on the nature of an individual problem.

Our computational experience shows that tailoring the block labeled AGMT-VAR in Fig.6.1 (the subroutine which augments the partial solution when (3) occurs)

---

*The backtrack procedure was first proposed by Glover[33], A detailed explanation can also be found in [28][48], and will not be described in this thesis.

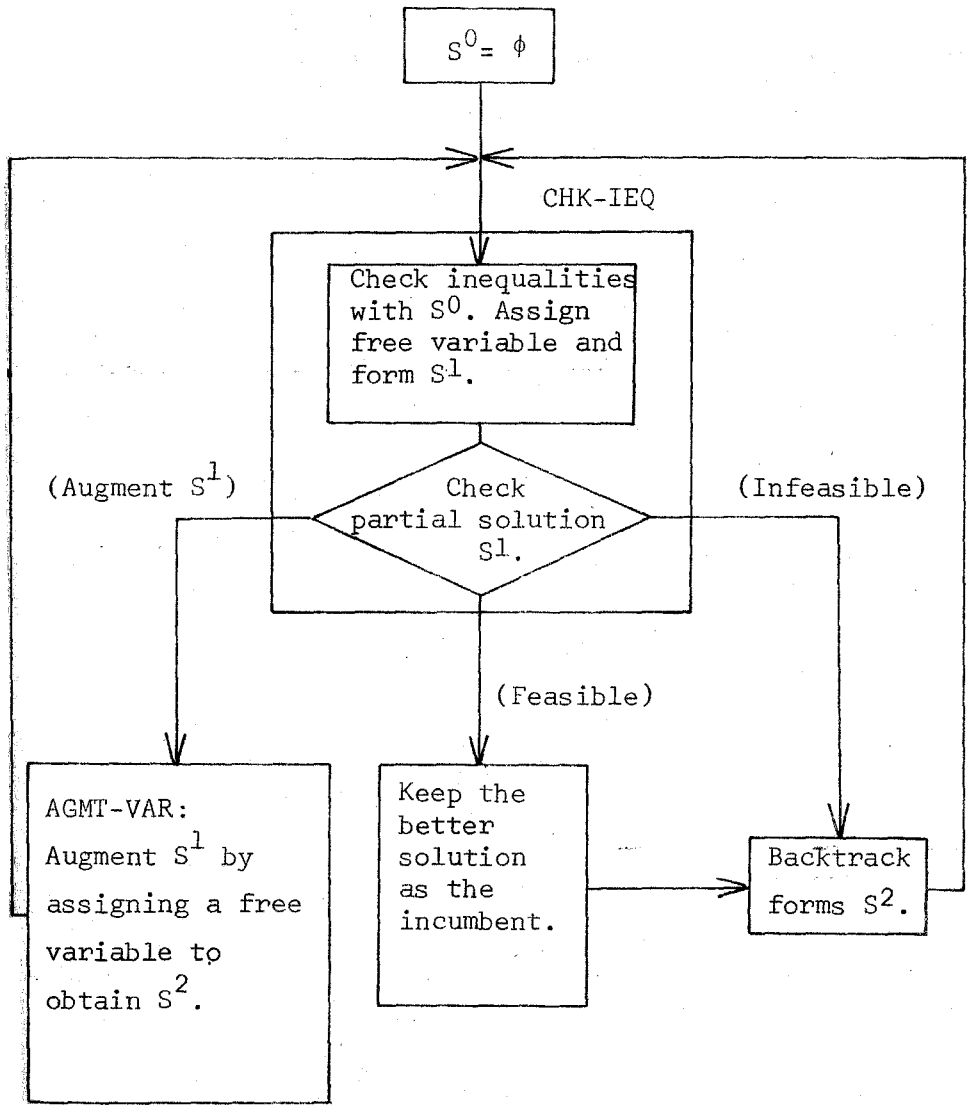


Fig.6.1. Implicit enumeration algorithm

to a given particular problem speeds up the convergence. Some aspects of our AGMT-VAR tailored to NOR gate network synthesis will be discussed in Section 6.4.

To perform the computation more quickly, the use of a faster computer would be desirable. One of such possibilities is offered by the computer with the capability of parallel processing, as realized as Illiac IV. The improvement of implicit enumeration in this direction is discussed in [49].

6.3 NOR Network Synthesis

All the optimal feed-forward NOR gate networks of three variable switching functions are realized. Networks are optimum in the sense that the number of NOR gates in the networks is minimized first and the number of connections between gates and from external variables is minimized second.

Now, the basic configuration for a NOR feed-forward network is shown in Fig. 5.1. The problems we solved have three external variables x_1, x_2, x_3. Since only completely specified functions are considered, each input vectors $X = (x_1, x_2, x_3)$ is numbered as $x^{(j)}$, $j = 1, 2, \ldots, 8$, from $x^{(1)} = (000)$ through $x^{(8)} = (111)$.

Inequalities for characterizing a feed-forward network with R NOR gates which realizes the function f is as follows (section 5.2):

$$-\sum_{\ell=1}^{3} v_\ell^k x_\ell^{(j)} - \sum_{i=1}^{k-1} \rho_{ik}^{(j)} \geq 0 - U\,(1 - P_k^{(j)})$$

$$\sum_{\ell=1}^{3} v_\ell^k x_\ell^{(j)} + \sum_{i=1}^{k-1} \rho_{ik}^{(j)} \geq 1 - U\,P_k^{(j)} \qquad (6.2)$$

$$k = 1, 2, \ldots, R-1$$
$$j = 1, 2, \ldots, 8.$$

where U is a sufficiently large positive number such that the upper inequality is non-binding if $P_k^{(j)} = 0$ and

the lower inequality is non-restrictive if $P_k^{(j)} = 1$.

For the last gate (the R-th gate),

$$- \sum_{\ell=1}^{3} v_\ell^R x_\ell^{(j)} - \sum_{i=1}^{R-1} \rho_{iR}^{(j)} \geq 0 \text{ if } f(x^{(j)}) = 1$$

$$\sum_{\ell=1}^{3} v_\ell^R x_\ell^{(j)} + \sum_{i=1}^{R-1} \rho_{iR}^{(j)} \geq 1 \text{ if } f(x^{(j)}) = 0 \qquad (6.3)$$

$$j = 1, 2, \ldots, 8.$$

Also in place of the relation $\rho_{ik}^{(j)} = \phi_{ik} P_i^{(j)}$, we have the following inequalities:

$$- P_i^{(j)} - \varphi_{ik} + \rho_{ik}^{(j)} \geq - 1$$

$$P_i^{(j)} + \varphi_{ik} - 2\,\rho_{ik}^{(j)} \geq 0 \qquad (6.4)$$

$$k = 2, 3, \ldots, R$$
$$i = 1, 2, \ldots, k-1$$
$$j = 1, 2, \ldots, 8.$$

The procedure used in Procedure 1 of Section 5.4 with the objective function,

$$\sum_{k=1}^{R} \left(\sum_{i=1}^{k-1} \varphi_{ik} + \sum_{\ell=1}^{3} v_\ell^k \right) \qquad (6.5)$$

is employed for the generation of all the optimal networks.

Seemingly the implicit enumeration algorithm has a tendency that the smaller the region of all solutions, the faster the convergence. Hence our effort was directed to preclude unnecessary solutions by adding extra inequalities so that the solution region becomes smaller without prohibiting any necessary optimal solutions. These additional inequalities essentially consist of two types; one is to preclude redundant network configurations and the other is to partially suppress the geometrically equivalent networks (i.e., those with permuted gates). They are listed in the following. Proofs are omitted.

(1) A NOR gate (B in Fig.6.2(a)) which has only one input from another gate A with only one output, and which is not the output gate is not permitted in an optimal network, because the same function can be realized

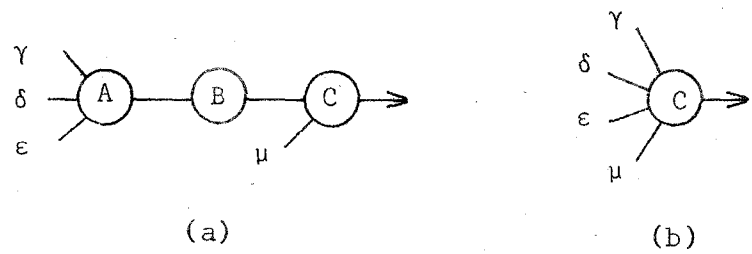


Fig. 6.2. Cascaded connections.

with two fewer gates, as shown in Fig. 6.2(b). Hence each gate except the last has at least one input from

the external variables or at least two inputs from the other gates, which is expressed by an inequality:

$$2 \sum_{\ell=1}^{3} v_{\ell}^{k} + \sum_{i=1}^{k-1} \varphi_{ik} \geq 2 \qquad (6.6)$$

$$k = 1, 2, \ldots, R-1.$$

Note that this condition does not hold in general when the fan-ins restiction is imposed.

(2) Consider any subnetwork which has only one gate which has outputs to gates not in the subnetwork (Fig.6.3).

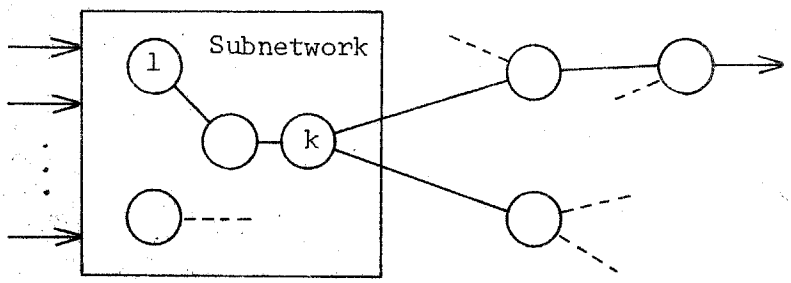


Fig.6.3. Ordering of inputs.

Assume that the subnetwork consists of gate 1 through k. Let the k-th gate is the output gate of this subnetwork. Then we can order the connections to the k-th gate from the other gates in the subnetwork such that $\phi_{1k} \leq \phi_{2k} \leq \cdots \leq \phi_{k-1,k}$. In other words, inequalities:

$$\varphi_{1k} \le \varphi_{2k} + \sum_{i=1}^{k-1} \sum_{j=k+1}^{R} \varphi_{ij},$$

$$\cdots$$

$$\varphi_{k-2,\,k} \le \varphi_{k-1,k} + \sum_{i=1}^{k-1} \sum_{j=k+1}^{R} \varphi_{ij} \qquad (6.7)$$

$$k = 3, 4, \ldots, R,$$

where the double sum $\Sigma\Sigma$ is added in each inequality because it is assumed that the subnetwork has outputs only from one gate (in other words if all the gates in the subnetwork except the k-th gate have no output to the outside of the subnetwork, i.e., if $\Sigma\Sigma\phi_{ij} = 0$, then (6.7) yields $\phi_{1k} \le \phi_{2k} \le \cdots \le \phi_{k-1,k}$.).

In the above discussion the subnetwork was assumed to consist of gate 1 through k. The extension to the general case in which the gates are not consecutively numbered starting from 1 is possible.

(3) Suppose that three gates are connected as shown in Fig.6.4, where the j-th gate has no outputs except ϕ_{jk}. It is easily proved that if all of ϕ_{ij}, ϕ_{ik}, ϕ_{jk} are 1, the network is not optimal, thus introducing inequalities:

$$\varphi_{ij} + \varphi_{ik} + \varphi_{jk} \le 2 + \sum_{\substack{t=j+1 \\ t \ne k}}^{R} \varphi_{jt} \qquad (6.8)$$

$$i < j < k \le R.$$

Even if the i-th gate in Fig.6.4 is replaced by an external variable x_ℓ, the above property is still true.

Then from (6.8) we obtain:

$$v_{\ell}^{j} + v_{\ell}^{k} + \varphi_{jk} \leq 2 + \sum_{\substack{t=j+1 \\ t \neq k}}^{R} \varphi_{jt} \qquad (6.9)$$

$$j < k \leq R$$

$$\ell = 1, 2, 3.$$

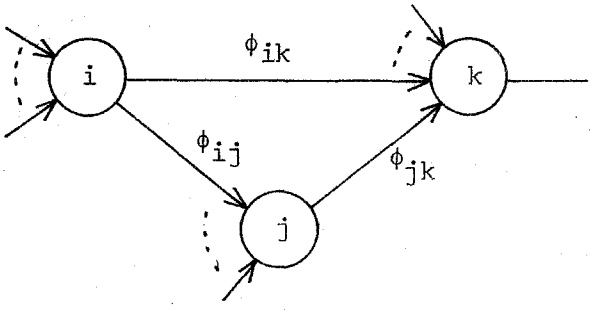


Fig.6.4. Triangular connections.

(4) This condition is an extension of case (3). Consider any subnetwork which has no outputs except those to the k-th gate and where the i-th gate and the k-th gate is connected, as shown in Fig.6.5. Assume that the subnetwork consists of the (i+1)st to (k-1)st gates. Then the connections from the i-th gate to the subnetwork are all redundant and therefore can be deleted. This condition is written by an inequality:

$$\sum_{j=i+1}^{k-1} \varphi_{ij} \leq 0 + U \left(\sum_{h=1}^{k-i-1} \sum_{j=k+1}^{R} \varphi_{i+h,j} + (1-\varphi_{ik}) \right) \qquad (6.10)$$

$$i = 1, 2, \ldots, R-2$$
$$k = i+2, \ldots, R,$$

where $U \geq \sum_{j=i+1}^{k-1} \phi_{ij}$ for all i and k.

Even if the i-th gate is replaced by an external variable, the above property is still true. In this case the inequality is:

$$\sum_{j=i+1}^{k-1} v_\ell^j \leq 0 + U \left(\sum_{h=1}^{k-i-1} \sum_{j=k+1}^{R} \varphi_{i+h,j} + (1-v_\ell^k) \right) \tag{6.11}$$

$$\ell = 1, 2, 3$$
$$i = 1, 2, \ldots, R-2$$
$$k = i+2, \ldots, R,$$

where $U \geq \sum_{j=i+1}^{k-1} v_\ell^j$ for all i and k.

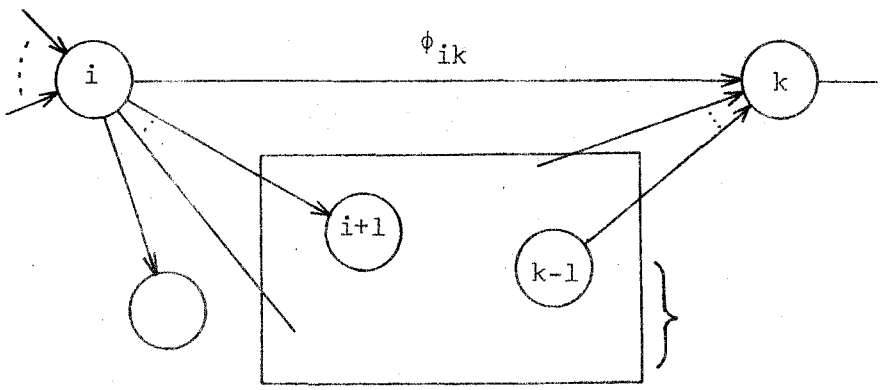


Fig.6.5. Generalized triangular connections.

In the above discussion it was assumed that the subnetwork consists of the gates of the consecutive numbers (i.e., from the (i+1)st to (k-1) st gates) but an extension to the more general case in which the subnetwork consists of gates of non-consecutive numbers is possible.

(5) A certain geometrical symmetry is also investigated. For example, Fig.6.6 shows three gates connected to the last gate.

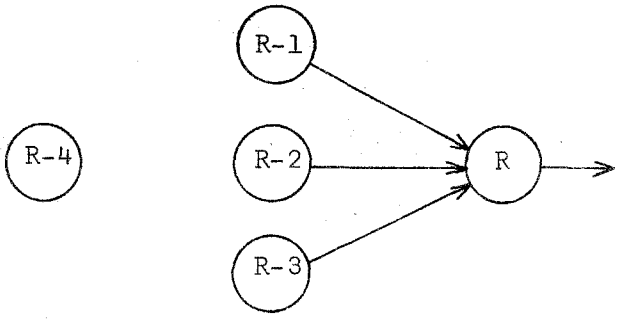


Fig.6.6. Symmetry of a subnetwork

In this configulation, it makes no difference to which of three gates the (R-1), (R-2), and (R-3), the (R-4)th gate is connected. Hence we can impose an ordering such that

$$\varphi_{R-4,R-1} \geq \varphi_{R-4,R-2} \geq \varphi_{R-4,R-3}$$

in order to preclude part of the gate configurations which are equivalent by permuting gates. This particular condition is represented by,

$$\phi_{R-4,R-3} \leq \phi_{R-4,R-2} + (1 - \phi_{R-3,R}) + (1 - \phi_{R-2,R}) + (1 - \phi_{R-1,R})$$

$$\phi_{R-4,R-2} \leq \phi_{R-4,R-1} + (1 - \phi_{R-3,R}) + (1 - \phi_{R-2,R}) + (1 - \phi_{R-1,R}). \qquad (6.12)$$

Similar types of symmetry conditions are extensively considered and a number of such inequalities are employed in the actual computation. However, the details of each type is not given here.

(6) If the connection between the i-th and (i+1)st gates is known to be 0, these two gates are geometrically equivalent and their output connections can be ordered.

Hence we can first order their connections to the last gate as

$$\varphi_{i,R} \leq \varphi_{i+1,R}. \qquad (6.13)$$

If it turns out that $\phi_{iR} = \phi_{i+1,R}$ then the connections to the (R-1)st gate can be ordered as

$$\varphi_{i,R-1} \leq \varphi_{i+1,R-1}. \qquad (6.14)$$

This argument continues until $\phi_{ik} \neq \phi_{i+1,k}$ $(k > i+1)$ eventually holds. After that, no ordering is permitted. This sequential condition is expressed by the inequality:

$$\sum_{j=2}^{R-i} 2^{j-1} \varphi_{i,i+j} \leq \sum_{j=2}^{R-i} 2^{j-1} \varphi_{i+1,i+j} + U \varphi_{i,i+1}, \qquad (6.15)$$

$$i = 1, 2, \ldots, R-2.$$

(7) Each gate has at least one output, because the network is assumed to have exactly R gates. This condition is expressed by

$$\sum_{j=k+1}^{R} \varphi_{kj} \geq 1 \qquad (6.16)$$

$$k = 1, 2, \ldots, R-1.$$

6.4 Computational Results of NOR Network Synthesis

With all the inequalities prepared in Section 6.3, integer programming problems are formulated.

The size of the problem is given in Table 6.1, both with a selected subset of the additional inequalities and without them.

These problems were run on the IBM 360/75I computer, following Procedure 1 of Section 5.4 with the objective function (6.4).

Our problems may be characterized by the following remarks:

(1) There are more inequalities than variables. Therefore the solution region is usually very small. In fact, many problems were found to be infeasible.

(2) The non-zero coefficients in the inequalities are fairly sparse. This feature was extensively utilized in our computer program of the implicit enumeration algorithm[48] in order to speed up the computation.

(3) The objective function has only coefficients of 0 or 1. This also simplifies the algorithm and is fully utilized[48].

The implicit enumeration algorithm was used on the NOR synthesis problem and the results are given in Table 6.2. The algorithm was set to enumerate all optimal feasible solutions. It also assumed no fan-ins and

		Without additional inequalities		With additional inequalities
R	No. of integer programming variables	Total no. of inequlities	% of non-zero coefficients*	Total no. of inequalities
2	23	40	14.6	-
3	52	88	7.1	-
4	90	152	4.3	169
5	137	232	2.9	265
6	193	328	2.1	415
7	258	440	1.6	716

Table 6.1. Size of NOR network formulation

*For the function f(X) = 0. For other functions, the sizes of figures are almost equal to those in the table.

R	No. of func's with exactly R NOR's.	Feasible				Infeasible	
		Average comp. time/function (seconds)	Average no. of iterations per function*	lst feasible solution (% of tatal iterations)	lst optimal solution (% of tatal iterations)	Average comp. time/function (seconds)	Average no. of ite. per function*
1	3	Exhausted by hand.					
2	5						
3	8	.275	4	-	-	.075	2
4	17	.90	30	-	-	.49	27
5	23	3.59	104	23.19	35.92	1.99	73
6	15	42.26	954	11.15	58.48	30.52	843
7	6	982.00	15,908	23.81	64.40	-	-

Table 6.2. Computational results of multiple optimal NOR network synthesis without fan-ins and fan-outs restrictions.

*The number of iteration is defined as the number of times CHK-IEQ (see Fig.6.1) is entered.

fan-outs restrictions. The computation took approximately 110 minutes on the IBM 360/75I for all 80 representative functions of three variables.

The same problem was already solved by Hellerman[44] by actually exhausting all the network configurations and then finding the best network among them for each function. Of course, all the networks obtained by our method are identical to Hellerman's. However, the integer programming approach compares favorably with Hellerman's which consumed about 26 hours on the IBM 7090 computer.

We examined what are influential factors on the speed of our program. The effect of additional inequalities on speed-up was remarkable. Some problems tried for the R = 5 formulation was speeded up by the factor of 5 in computation time.

As explained so far many types of network restrictions such as fan-ins and fan-outs can easily be added to the synthesis problem in the form of inequalities, without the necessity of changing or complicating the program. This is one of the advantages of the integer programming formulation. When the fan-ins restriction is imposed, the restrictions of (6.6) can no longer be included since it is possible to have a gate with a single input from another gate.*

The excution times both with and without fan-ins and fan-outs restrictions, however, are on the average about the same for functions tested. (Although the computation time for each function is often different.)

---

*See the argument of Fig.6.9 in Section 6.5.

Also by simply changing one statement in the program, we can modify the algorithm so that a single optimal solution is obtained rather than exhausting all the optimal solutions. For 6 gate functions tested, the computation time decreased by 10 - 20 %.

Further considerable speed-up was gained by a more sophisticated modification of the implicit enumeration method by taking into account the physical structure of a network. This will be explained in the next section.

6.5 Improvement of NOR Network Synthesis by the All-Interconnection Network Formulation

In the following computer program, we use a somewhat different integer programming formulation based on the all-interconnection network* to characterize the feed-forward network, in addition to the reconstruction of AGMT-VAR. Also inequalities are added to break loops which may result by solving the integer program with the above all-interconnection network. This modification is employed because, we can preclude a large number of partial solutions which are equivalent simply by permuting gates and which are difficult to preclude by other means.

This all-interconnection network formulation turns out to be very powerful for speeding up the NOR network synthesis.

The AGMT-VAR in Fig.6.1 is accordingly modified such that free variables of a partial solution be assigned, starting with the free variables associated with gate R, and proceeding to gate R-1, then R-2, until reaching gate 1. This procedure also insures that there is no isolated subnetwork consisting of more than one gate. There may be various conceivable versions but we made the new AGMT-VAR based on Davidson's desirability order of gates, since it looked appropriate. This indicates the flexibility of our synthesis method by integer programming.

Recently Davidson[19] applied the "branch and bound" method** to the NAND network synthesis, without describing the network by inequalities.

---

*See Section 5.10B.

**See the next page.

An outline of his approach is as follows. The algorithm starts from the network of a single gate with its output specified to $f(x^{(j)})$, $j = 1, 2, \ldots, m$. To realize this f, input values to the gate are successively specified, inducing the addition of new connections or gates if required to satisfy the current input-output relation for NOR gates in the network. If we reach a network in which each gate satisfies the input-output relation as a NOR gate and the last gate realizes f, then the network is a NOR network realizing $f(x)$. During the execution of the algorithm, however, there usually occur several choices in proceeding one step forward. For example, to realize a certain input value to a gate, one new gate may be added, or one connection is made without increasing the number of gates. To exhaust all these possibilities and to obtain the optimal among them, Davidson adopted the branch and bound principle[60] tailored to this particular problem.

He achieved a remarkable reduction of computation time, by judiciously making use of the intrinsic properties of a NAND gate. He defined the types of gates and determined empirically the desirability order of the types of gates. By checking all possible partial networks with very elaborate procedures, the optimality of the obtained network is guaranteed.

---

*The implicit enumeration algorithm might be regarded as a synonym of the branch and bound method. But Davidson's algorithm and ours are quite different. Davidson's method is without inequalities, whereas ours is entirely based on the inequality manipulation.

The types of gates and the desirability order introduced by Davidson were incorporated with some modifications in our new AGMT-VAR. Our new computer program is more complex than the previous one. (The reduction of computation time was made at the expense of the complexity of the program.) It appears still simpler than Davidson's because the inequalities are still used and these inequalities provide simultaneously much information about the current partial solution without complicating program procedures (i.e., information about types of gates or covering condition which will be defined in the following).

The principle of this approach is based on the properties of a NOR gate that:

Property 1: If $P_k^{(j)} = 1$, then

$$\rho_{ik}^{(j)} = 0 \quad , \quad v_\ell^k \, x_\ell^{(j)} = 0,$$

for all $i \neq k$ and $\ell = 1, 2, 3$.

Property 2: $P_k^{(j)} = 0$, then there exist at least one i ($i \neq k$) or ℓ ($1 \leq \ell \leq 3$) such that

$$\rho_{ik}^{(j)} = 1 \text{ or } v_\ell^k \, x_\ell^{(j)} = 1.$$

As defined before, $P_k^{(j)}$ is the output value of the k-th gate for the j-th input vector, $\rho_{ik}^{(j)}$ is the input value of the k-th gate supplied from the i-th gate for the j-th input vector, and v_ℓ^k is the connection of the ℓ-th external variable to the k-th gate.*

*Note that during the computation the variables can take on 3 values 0, 1, and * (unassigned).

Given a partial solution S in the course of the implicit enumeration, we have a partially constructed NOR network which may or may not lead to an optimal network eventually. Our modification of the algorithm is simply the reconstruction of the subroutine AGMT-VAR (see Fig.6.1) such that it augments the partial solution with a variable selected accoding to the above property 2 of NOR gate in order to derive a reasonably good next parital solution from the current one. It is difficilt to know what is good and there is no proof that the following procedure gives a good solution. However, our computational experience showed that it was considerably better than the AGMT-VAR which had been designed for general use and explained in the earlier sections.

Now a few definitions are given. A gate is said to be isolated if it has no output connected to other gate in the current partial solution. In other words, if all ϕ_{ik}, $k \neq i$ are 0 or *, the gate i is isolated. (The i-th gate is currently isolated but could be connected assinging 1 to a free variable.) If $P_k^{(j)}$ is 0 in the current partial solution, let us associate with it one of the types which will be defined in the following. Let us define the types, "COV", "G*" and so on.*

These types are almost identical to Davidson's[19] but slightly different. And the number of types in this paper is fewer. G, VC*, GC*, C*C*, NWG, approximately correspond to Davidson's EXP, VAR, FCN, EXF, NF, respectively. ISL and others which will be defined later are new concepts.

COV; If there exist i or ℓ such that $\rho_{ik}^{(j)}$ or $x_{\ell}^{(j)} v_{\ell}^{k} = 1$ (i.e., COVered.).

G*; If there exists i such that $\phi_{ik} = 1$ and $P_i^{(j)} = *$ (G* stands for the Gate i with $P_i^{(j)}$ unassigned.)

VC*; If there exists ℓ such that $x_{\ell}^{(j)} = 1$ and $v_{\ell}^{k} = *$. (VC* stands for the Variable $x_{\ell}^{(j)} = 1$ with Connection being *)

GC*; If there exists i such that $P_i^{(j)} = 1$ and $\phi_{ik} = *$, when the gate i is not isolated (Gate i with $P_i^{(j)} = 1$ and Connection being *).

G*C*; If there exists i such that $P_i^{(j)} = *$ and $\phi_{ik} = *$, where the gate i is not isolated. (Gate i with $P_i^{(j)} = *$ and Connection being *)

NWG; If there exists i such that $P_i^{(j)} = *$ and $\phi_{ik} = *$, where the gate i is isolated (NWG stands for NeW Gate.).

Since each $P_k^{(j)}$ may satisfy more than one of the above conditions (i.e., each $P_k^{(j)}$ may be associated with more than one type), let us order these types by desirability as

$$\text{COV, G*, VC*, GC*, G*C*, NWG.} \qquad (6.18)$$

And the type of $P_k^{(j)}$ is defined as the most desirable one among those which satisfy the above definition, if $P_k^{(j)} = 0$ in the current partial solution.

The motivation for defining these is to find a gate or external variables which leads to Property 2 of the earlier discussion. If Property 2 is satisfied or equivalently, if the type of $P_k^{(j)}$ is COV, $P_k^{(j)}$ is said covered. In the desirability order (6.18), G*, for example, is more desirable than G*C* since it is more likely that the covering may be achieved by assigning 1 to * of the gate which was already connected.(i.e., no new gate added.) But it is rather difficult to see intuitively that G* is more desirable than VC*. (6.18) was determined empirically by Davidson such that the computation time is minimized. In this sense the order of desirability may be susceptible to the type of gates with which the network is to be synthesized. In general, it can be determined only by trial-error programming experiments.

Note that by our implicit enumeration algorithm, every partial solution automatically satisfies Property 1 because otherwise it is rejected when the check of the partial solution is done in CHK-IEQ of Fig.6.1.

As an auxiliary concept to the following definition of the type of partial network, let us define <u>types of gates</u>:

ISL; If the gate is isolated (ISoLated).

LTG; If the gate is not of ISL type and the outputs $P_k^{(j)}$ for all j's are 0 or *.(LaTently useful Gate in the sense that the covering condition may be met later.)

If a given gate k satisfies neither of the above definitions, then the least desirable type according to order (6.18) among the types of $P_k^{(j)}$ for all j's such that $P_k^{(j)} = 0$ is defined as the type of gate k.

If the types of all the gates in the current partial solution are ISL, LTG, or COV, then the current partial solution is feasible*, i.e., the resulting feasible network, is easily derived. If this is the case, the backtrack (Fig.6.1) procedure is entered after comparing this feasible (network) solution with the incumbent.

Let the desirability of types of gates be defined as

ISL, COV, LTG, G*, VC*, GC* G*C*, NWG.

Then the type of partial network is defined as the least desirable type of gate among all gates.

Fig.6.7 shows a partial network corresponding to a current partial solution. Each $P_k^{(j)} = 0$ is shown with its type for $x^{(j)}$ of Fig.6.8. For example, $P_3^{(1)}$ is of type GC* because by setting $\phi_{23} = 1$, $P_3^{(1)}$ can be covered. $P_3^{(2)}$ is of type G*C* because of $P_2^{(2)} = *$. The type of each gate is shown in Fig.6.7. Obviously, the type of this parital network is G*C* which is the type of gate 3.

Let us describe the procedure in the new AGMT-VAR. If the type of partial network is one of ISL, COV, LTG, the current partial solution is feasible because we use

---

*In our design procedure where the number of gates in a network starts at 1 and then increases, the feasible case occurs only when the types of all the gates are COV.

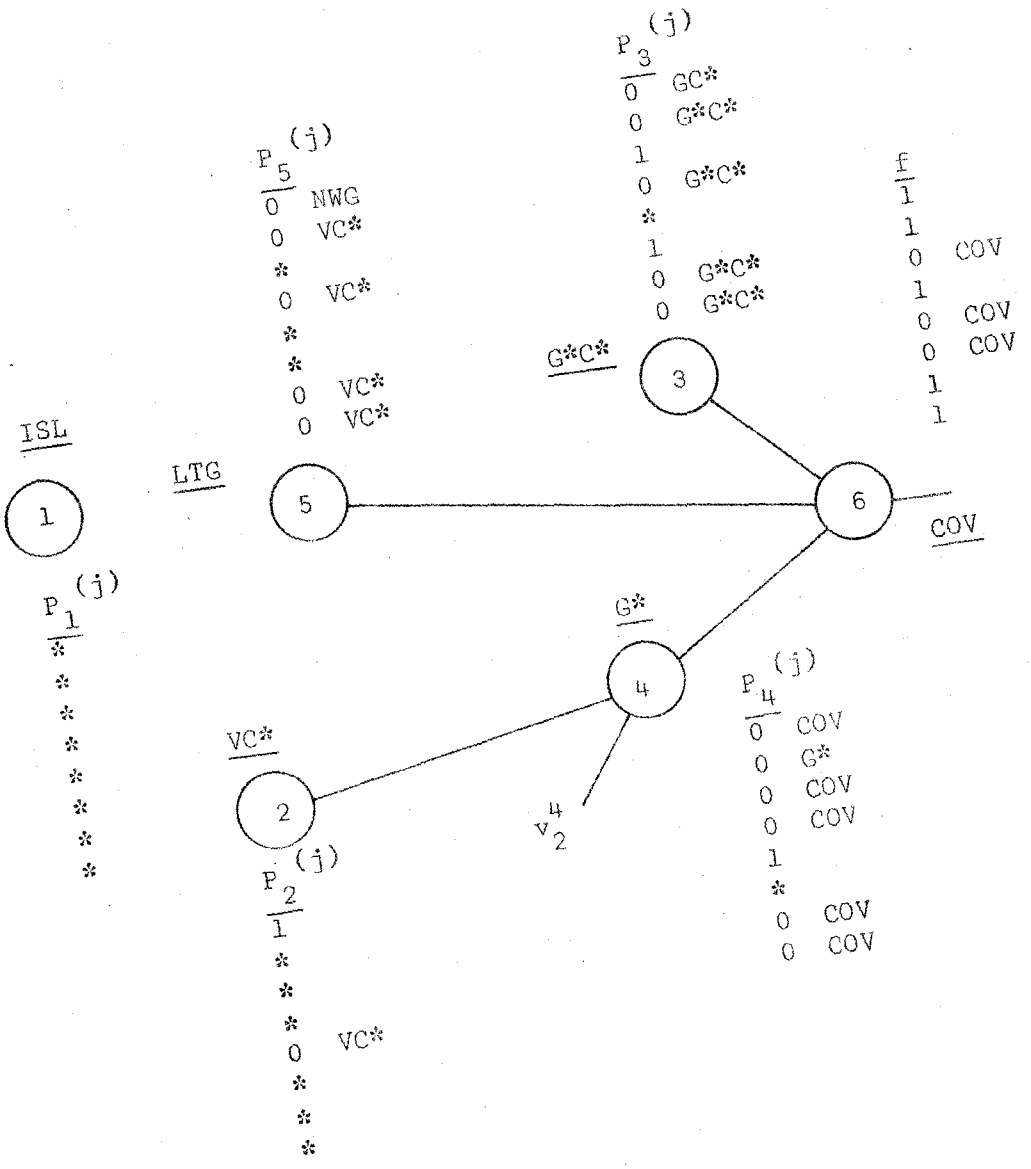


Fig.6.7. Example of types of $P_k^{(j)} = 0$, gates, and the partial network.

j	x_1	x_2	x_3
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Fig.6.8. Assignment of values to $x^{(j)}$.

Procedure 1 in Section 5.4. Otherwise let us identify $P_K^{(j)}$ which defines the type of this partial network. In the definition of the types G*, VC*, GC*, G*C*, and NWG, free variables are associated with $P_k^{(j)}$ such that if these free variables are set to 1, $P_k^{(j)}$ is covered (e.g., in the case of VC*, $P_k^{(j)}$ is covered if v_ℓ^k is set to 1, i.e., if the corresponding free variable is set to 1.). This free variable is then specified so that $P_k^{(j)}$ is covered. For example, Fig.6.7 shows that $P_3^{(2)}$ is identified according to the type of partial network. $P_3^{(2)}$ is of type G*C* and can be covered by connecting gates 2 and 3, i.e., by setting $\phi_{23} = 1$*, $P_2^{(1)} = 1$. Others are similarly treated; an appropriate $\rho_{ik}^{(j)}$ in the case of G*, GC*, G*C* types or an appropriate v_ℓ^k in the case of VC* is set to 1. However, the type NWG is treated differently. If a gate is of type NWG, the isolated gate which has the largest gate number is identified. Let it be γ. Then $\rho_{\gamma k}^{(j)}$ is set to 1 where $P_k^{(j)}$ defines the type of partial network. Setting $\rho_{\gamma k}^{(j)}$ to 1 will force $\phi_{\gamma k}$ and $P_\gamma^{(j)}$ to 1, thus covering $P_k^{(j)}$. However, if $\rho_{\gamma k}^{(j)} = 0$ (i.e., $\rho_{\gamma k}^{(j)}$ is not a free variable), all solutions with anyone of isolated gates connected to gate k have been investigated and do not need to be checked again.

---

*In actual programming, this is done by augmenting the current partial solution by $\rho_{23}^{(2)} = 1$ rather than $\phi_{23} = 1$ and $P_2^{(2)} = 1$. Subsequently ϕ_{23} and $P_2^{(2)}$ are set to 1 when the partial solution undergoes CHK-IEQ.

This follows from the fact that gate γ and any other isolated gate are equivalent with respect to the partial network (i.e., non-isolated part) since any isolated gate can be connected to any other gate. This approach eliminates the permutation of gate labeling which is inevitable with the feed-forward network. Of course, if we discover that $\rho_{\gamma k}^{(j)}$ was already set to 0, then we can simply abandon the current partial solution and go to the backtrack procedure.

A comment is given on the case in which the type of the partial network is VC*. If the type is VC*, sometimes more than one external variable can be connected. In our algorithm, among all possible external variables, the external variable which covers the largest number of $P_k^{(j)} = 0$ not yet covered is connected to the gate.

This new AGMT-VAR is used in place of AGMT-VAR shown in Fig.6.1 and assigns a free variable to 1 according to the type of the partial network. Then CHK-IEQ is applied as before. Other part of algorithm are exactly the same as the general case discussed in the earlier sections, except that the following objective bound check is added to AGMT-VAR to further speed up the computation.

Given a partial solution, a lower bound of the objective function value is estimated according to the rules based on the following properties. If this bound exceeds the objective value of the incumbent, the current partial solution can not give any better solution than that and accordingly it is discarded. (The rest of

computation for the current partial solution is skipped and backtrack is immediately applied). The bound estimation is programmed by using the following properties of the network:

(1) Exactly R gates are assumed to be used in the network. In other words, each gate has at least one input connection and at least one output connection.

(2) According to condition (1) of Section 6.3, each gate has either at least one input connection from external variable or at least two input connections from other gates.

(3) The number of gates each of which is solely devoted to expressing $\bar{x}_i$ (i.e., a gate has only one input which is an external variable x_i) is at most three.

(4) If the type of gate is GC*, G*C*, or NWG, the gate needs at least one more connection from another gate, as seen from the definitions of these types.

(5) If the type of partial network is VC*, the gate whose type defines the type of partial network needs at least one more connection from an external variable. If this single external variable does not cover all the $P_k^{(j)}$ which are of type VC*, we need to add at least one more external variable.

This bound estimation is included in AGMT-VAR and

whenever it demonstrates that the current partial solution can not give any better feasible solution than the incumbent the backtrack procedure is performed immediately.

If the fan-ins and fan-outs restrictions are considered, the rules based upon the above properties (2) and (3) must be modified. All other rules are unchanged. For example, let us assume that each gate has maximum fan-ins and fan-outs of 3. Let us examine Fig.6.9.

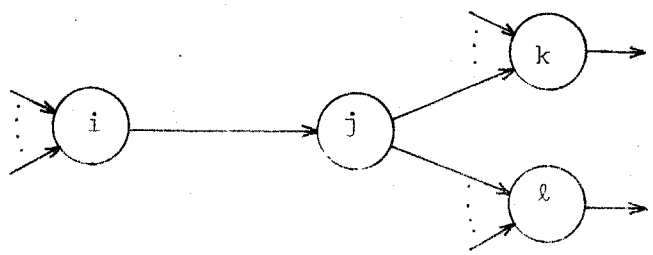


Fig.6.9 Modification due to fan-ins and fan-outs restrictions.

The single input to gate j is not allowed only if

$$\sum_{t=1}^{3} (v_t^i + v_t^k) + \sum_{\substack{t=1 \\ t\neq i}}^{R-1} \varphi_{ti} + \sum_{\substack{t=1 \\ t\neq j \\ t\neq k}}^{R-1} \varphi_{tk} \leq 3 \qquad (6.20)$$

$$\text{and} \quad \sum_{t=1}^{3} (v_t^i + v_t^\ell) + \sum_{\substack{t=1 \\ t \neq i}}^{R-1} \varphi_{ti} + \sum_{\substack{t=1 \\ t \neq j \\ t \neq \ell}}^{R-1} \varphi_{tk} \leq 3. \qquad (6.21)$$

Property (3) also must be modified.

6.6 Computational Results of Improved Algorithm

With all these modifications discussed in the previous section added, all 15 functions which can be realized with 6 NOR gates were tested on the R = 6 formulation. The improvement was remarkable. The average number of iterations per function is 136.3 and the average computation time for each function is 4.99 seconds which is favorably compared with the result in Table 6.2 in Section 6.4 run with the general purpose AGMT-VAR, in which 954 iterations and 42.26 seconds on the average were necessary for each function.

Similarly, the odd parity function $x_1 \oplus x_2 \oplus x_3$ which requires 8 NOR gates when the fan-ins and fan-outs are limited to 3 was solved. In this case, the modifications of the bound estimation arisen from the fan-ins and fan-outs restriction is incorporated, as explained in Section 6.5.

Since there is only one function to be solved, the structure of an optimal network for the function $x_1 \oplus x_2 \oplus x_3$ is taken into consideration. In other words, this function is symmetric in all the three variables x_1, x_2, x_3 and accordingly the connections from x_ℓ to the 7th gate are ordered as follows.

$$v_3^7 \geq v_2^7 \geq v_1^7 \qquad (6.22)^*$$

*It can be easily shown that the last gate (the 8-th gate) has no external variables connected, if a given function can not be expressed as

$$f = \bar{x}_{i_1} \bar{x}_{i_2} \ldots \bar{x}_{i_k} g(x).$$

and if $v^7_{\ell+1} = v^7_{\ell}$, then

$$v^6_{\ell+1} \geq v^6_{\ell} \qquad \ell = 1, 2. \quad (6.23)$$

((6.23) could be continued to the gate numbers lower than 6. But the continuation was not incorporated in our program.) Also two connections ϕ_{68} and ϕ_{78} are assumed to be 1. This is guaranteed by the fact that the networks which are constructed by adding one gate to the output of optimal 7 NOR gate networks of $\overline{x_1 \oplus x_2 \oplus x_3}$, give no better realization of $x_1 \oplus x_2 \oplus x_3$ than those obtained by solving the 8 gate formulation.

The problem has 395 variables and 1012 inequalities including additional inequalities. All optimal solutions are shown in Fig.6.10. The network (c) of Fig.6.11 is the same as that found by Taniguchi et al.[96]

The computation took 4822 iterations with network (a) occuring at the 1368-th iteration, network (b) at 1647-th iteration, and solution (c) at the 3052-th iteration. The total computation took less than 6 minutes and 30 seconds on the IBM 360/75I computer. When the program was modified so that only one of the optimal solutions is to be found, the time was reduced to 5 minutes and 15 seconds. These computation times are a considerable reduction with respect to the results of Table 6.2 in Section 6.4. Also, Hellerman's approach would need impractically large amount of computation time such as

---

*Recently, further 2-3 times speed up was attained by improving the implicit enumeration algotithm.(Private communication with S. Muroga.)

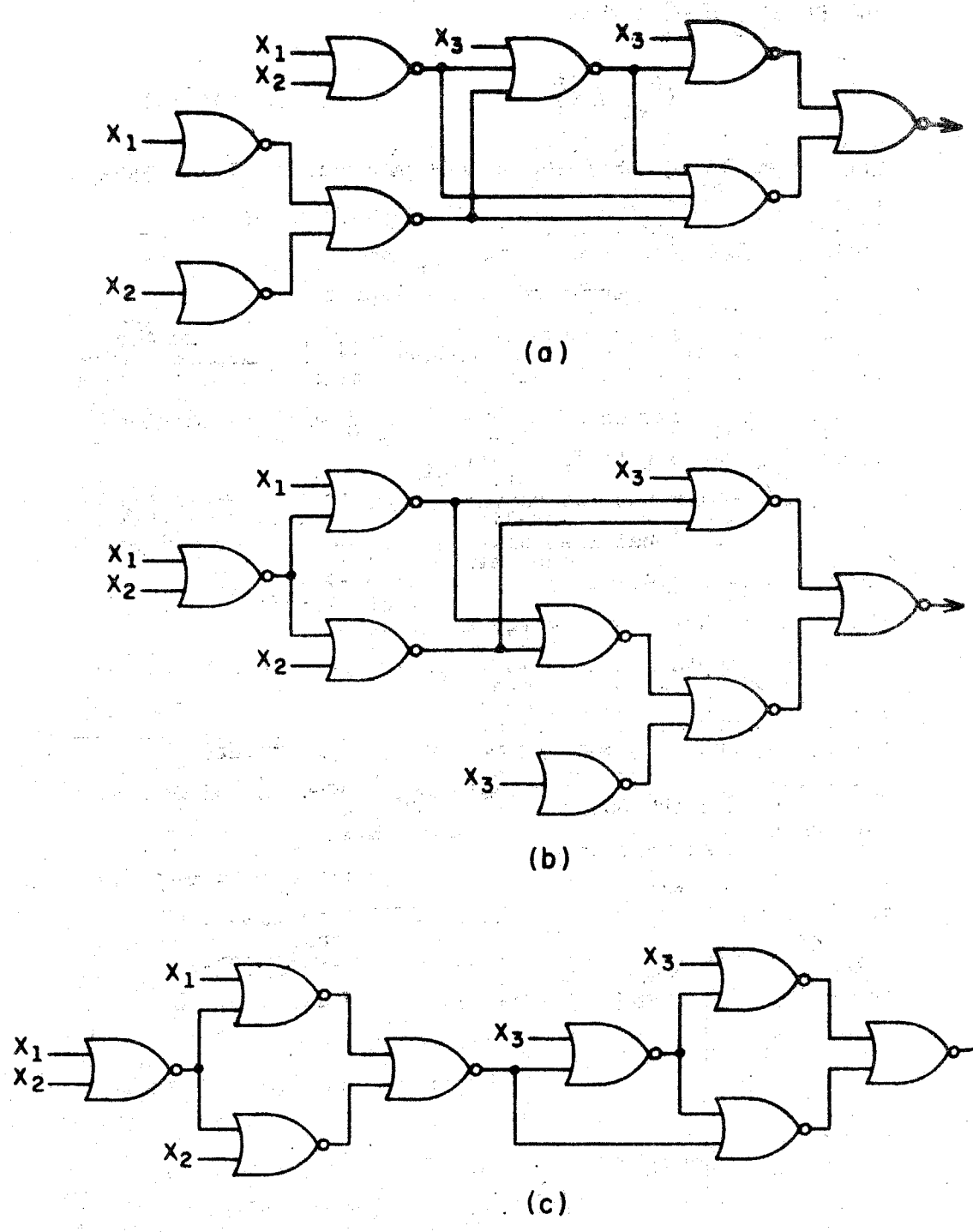


Fig.6.10. All optimal networks for $x_1 \oplus x_2 \oplus x_3$.

over a few thousand hours of IBM 7090.

The reduction of the computation is due to the desirability order of types and the all-interconnection formulation. The all-interconnection network formulation appears to contribute more to the reduction than the desirability order. The desirability order probably has to be changed when different types of gates are to be used.

Our computation times are about the same as Davidson's, though exact comparison is very difficult because computers used are different and moreover simple programming gimmicks may further make changes in computation time.

As a nature of branch and bound approach, it does not assume in advance the number of gates and connections needed to realize a given function f. It sharply contrasts to the integer programming approach in which all inequalities are generated for the assumed number of gates, R. Hence the branch and bound method will find the first feasible solution quickly, though it may use an excess number of gates compared with the minimum number. The integer programming method reaches the first feasible solution which has the minimum number of gates, via the steps for $R = 1, 2, \ldots$.

Probably one of the advantages of our approach is the ease of programming effort. Since we can fully use the infromation associated with variables of the inequalities, the procedure to determine the types of $P_k^{(j)}$, gates and eventually the partial network is fairly straightforward

and simple. Also Property 1 of the NOR gate is automatically taken care of by the CHK-IEQ routine, thus eliminating the procedure for checking this property. Another advantage is the fixed amount of storage needed throughout the computation. In Davidson's "branch and bound algorithm" the amount of storage often grows as the computation proceeds, to memorize the history of all choices and backtracks. Otherwise additional programming must be done in order to regenerate needed information or to use secondary storage.

Because of great improvement of the all-interconnection network formulation over the approach of Section 6.3, improvement in computation time even in the case of multiple-output network design also can be expected with the all-interconnection network formulation.

The accurate comparison of the exhaustive method, Davidson's and ours, in terms of the program complexity, computational efficiency, ease of programming and all other aspects, is difficult at this stage, because a number of factors should be considered for the comparison and the data gained so far is not sufficient. Here we tried this new approach to show the flexibility of the implicit enumeration algorithm. In other problems also, it is quite likely that we can achieve a great improvement by considering the intrinsic properties of the problem and incorporating appropriate modifications of the algorithm. Some examples in this respect will be found elsewhere[48].

6.7 NOR-AND Network Synthesis

One of the important advantages of integer programming formulation is that we can solve a wide variety of problems by simply changing the set of inequalities. Networks composed of NOR and AND gates are represented in inequalities in Section 5.9. Of course the optimal networks are also interesting in their own right. All the optimal networks for 80 switching functions of three variables are tabulated under the same optimality criterion as NOR networks.

The basic set of inequalities describing a feed-forward network of R gates is repeated in the following. For k = 1, 2, ..., R-1, (See Section 5.9)

$$
\begin{aligned}
&\sum_{\ell=1}^{3} v_\ell^k x_\ell^{(j)} + \sum_{i=1}^{k-1} \rho_{ik}^{(j)} \geq \sum_{\ell=1}^{3} v_\ell^k \\
&\qquad + \sum_{i=1}^{k-1} \phi_{ik} - U(1 - P_k^{(j)}) - U\theta_k \ , \\
&-\sum_{\ell=1}^{3} v_\ell^k x_\ell^{(j)} - \sum_{i=1}^{k-1} \rho_{ik}^{(j)} \geq -\sum_{\ell=1}^{3} v_\ell^k \\
&\qquad - \sum_{i=1}^{k-1} \phi_{ik} - UP_k^{(j)} - U\theta_k + 1 \ , \\
&-\sum_{\ell=1}^{3} v_\ell^k x_\ell^{(j)} - \sum_{i=1}^{k-1} \rho_{ik}^{(j)} \geq 0 - U(1 - P_k^{(j)}) \\
&\qquad - U(1 - \theta_k) \ , \qquad (6.24) \\
&\sum_{\ell=1}^{3} v_\ell^k x_\ell^{(j)} + \sum_{i=1}^{k-1} \rho_{ik}^{(j)} \geq 1 - UP_k^{(j)} \\
&\qquad - U(1 - \theta_k) \ , \\
&\qquad j = i, 2, \ldots, 8.
\end{aligned}
$$

It may be easily seen that the first two inequalities are for an AND gate and the others for a NOR gate, depending on the value of θ_k associated with these inequalities.

For the last gate, if $f(x^{(j)}) = 1$,

$$\sum_{\ell=1}^{3} v_\ell^R x_\ell^{(j)} + \sum_{i=1}^{R-1} \rho_{iR}^{(j)} \geq \sum_{\ell=1}^{3} v_\ell^R + \sum_{i=1}^{R-1} \phi_{iR} - U\,\theta_R$$

$$-\sum_{\ell=1}^{3} v_\ell^R x_\ell^{(j)} - \sum_{i=1}^{R-1} \rho_{iR}^{(j)} \geq 0 - U\,(1-\theta_R) \qquad (6.25)$$

and if $f(x^{(j)}) = 0$,

$$-\sum_{\ell=1}^{3} v_\ell^R x_\ell^{(j)} - \sum_{i=1}^{R-1} \rho_{iR}^{(j)} \geq -\sum_{\ell=1}^{3} v_\ell^R - \sum_{i=1}^{R-1} \phi_{iR} + 1 - U\,\theta_R$$

$$\sum_{\ell=1}^{3} v_\ell^{(j)} x_\ell^{(j)} + \sum_{i=1}^{R-1} \rho_{iR}^{(j)} \geq 1 - U(1-\theta_R) \qquad (6.26)$$

for $j = 1, 2, \ldots, 8$.

The relation $\rho_{ik}^{(j)} = \phi_{ik} P_i^{(j)}$ is expressed by

$$-P_i^{(j)} - \varphi_{ik} + \rho_{ik}^{(j)} \geq 1$$

$$P_i^{(j)} + \varphi_{ik} - 2\,\rho_{ik}^{(j)} \geq 0$$

$$k = 2, 3, \ldots, R$$

$$i = 1, 2, \ldots, k-1 \qquad (6.27)$$

$$j = 1, 2, \ldots, 8.$$

Additional inequalities are also incorporated to reduce the computation time. All of those additional inequalities which were used in the all NOR network case are included except the geometrical symmetry restrictions given by (5) of Section 6.3. In addition, the following two types of inequalities were added.

(1) Input connections: each AND gate has at least two input connections. This constraint is given by

$$\sum_{\ell=1}^{3} v_{\ell}^{k} + \sum_{i=1}^{k-1} \rho_{ik} \geq 2 - \theta_k \qquad (6.28)$$

$$k = 1, 2, \ldots, R.$$

(2) Triangular connections: Again consider three gates connected together as shown in Fig.6.11. Contrary to the NOR network case, if either of the gates j and k is AND gate, at least one of the three connections ϕ_{ij}, ϕ_{ik}, ϕ_{jk} must be 0

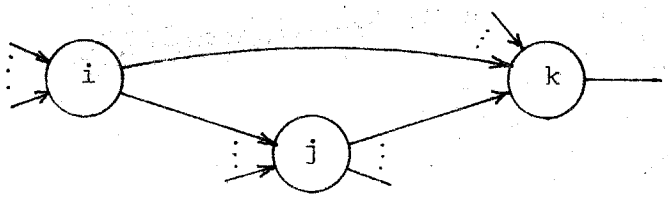


Fig.6.11. Triangular connection.

even if the j-th gate has outputs other than ϕ_{jk}. This condition is given by

$$\varphi_{ij} + \varphi_{jk} + \varphi_{ik} \leq 2 + \theta_j$$

$$\varphi_{ij} + \varphi_{jk} + \varphi_{ik} \leq 2 + \theta_k \qquad (6.29)$$

$$i < j < k \leq R.$$

When the i-th gate is replaced by an external variable, the followings will result.

$$v_\ell^j + v_\ell^k + \varphi_{jk} \leq 2 + \theta_j$$

$$v_\ell^j + v_\ell^k + \varphi_{jk} \leq 2 + \theta_k \qquad (6.30)$$

$$j < k \leq R$$

$$\ell = 1, 2, 3.$$

Together with these additional inequalities (some of additional inequalities based on the properties which have been discussed are not incorporated because of their excessive number.), all the optimal NOR-AND combination networks for each function are solved. The entire computation took about 54 minutes on the IBM 360/75I, using a program which includes the general purpose AGMT-VAR. All functions are realized with not more than six gates. The size of each problem and computation time are listed in Tables 6.3 and 6.4. The computation time is graphically given in Fig.6.12 as well as the NOR network case.

The effect of additional inequalities was remarkable.

R	No. of functions realizable with R gates (representative functions of equivalent classes.)	Size of integer programming prob.			% of non-zero entries in basic inequalities*
		No. of variables	No. of basic inequalities	No. of total inequalities including additional inequalities	
3	15	55	128	135	8.37
4	29	94	208	287	5.22
5	14	142	304	450	3.60
6	5	199	416	657	2.65

Table 6.3. Statistics of optimal NOR-AND network formulation.

*For the function which is identically 1.

R	No. of feasible functions	Feasible		Infeasible	
		Average computation time/ function (seconds)	Average no. of iterations/ function	Average computation time/ function (seconds)	Average no. of iterations/ function
0	3	Exhausted by hand			
1	5				
2	9				
3	15	0.82	40	0.54	36
4	29	3.62	115	3.11	110
5	14	34.9	844	28.7	745
6	5	475.0	8734	-	-

Table 6.4. Computation time and number of iterations of NOR-AND network synthesis for all 80 functions of three variables.

For the R = 3 case the computation time was reduced 6 times by incoporating the additional inequalities; and for the R = 4 case, 18 times. Incorporation of other additional constraints such as the constraint that each AND gate should have at least one NOR gate connected to its output, will reduce the computation time further, though these were not actually tried.

Also contrary to NOR gate network, no effort was made to improve the computational efficiency of the integer programming algorithm. Of course a significant reduction of computation time can be expected by modifying the algorithm as discussed in Section 6.5.

A comparison of computation time with NOR gate networks is shown in Table 6.5 where the ratio of the computation time of the NOR-AND network case to that of NOR network case discussed in Section 6.4 is listed.

On the average a function can be realized by NOR-AND combination network with 0.85 fewer gates than that of the NOR network case. The number of connections is also reduced by 1.5 on the average. All the optimal networks for three variable function are tabulated in Appendix.

The tabulation also gives all the optimal networks by NAND-OR combination by the following procedure: (1) take the dual of a given function f, (2) find optimal networks for the dual function f^d by NOR-AND combination and (3) replace NOR by NAND, and AND by OR. These are optimal realizations of f by NAND-OR combination.

R	Computation time		Iterations	
	Feasible	Infeasible	Feasible	Infeasible
4	4.0	6.3	3.8	4.1
5	9.8	14.4	8.1	10.2
6	11.3	-	9.2	-

Table 6.5. Ratio of computation time and iterations.

(NOR-AND / NOR)

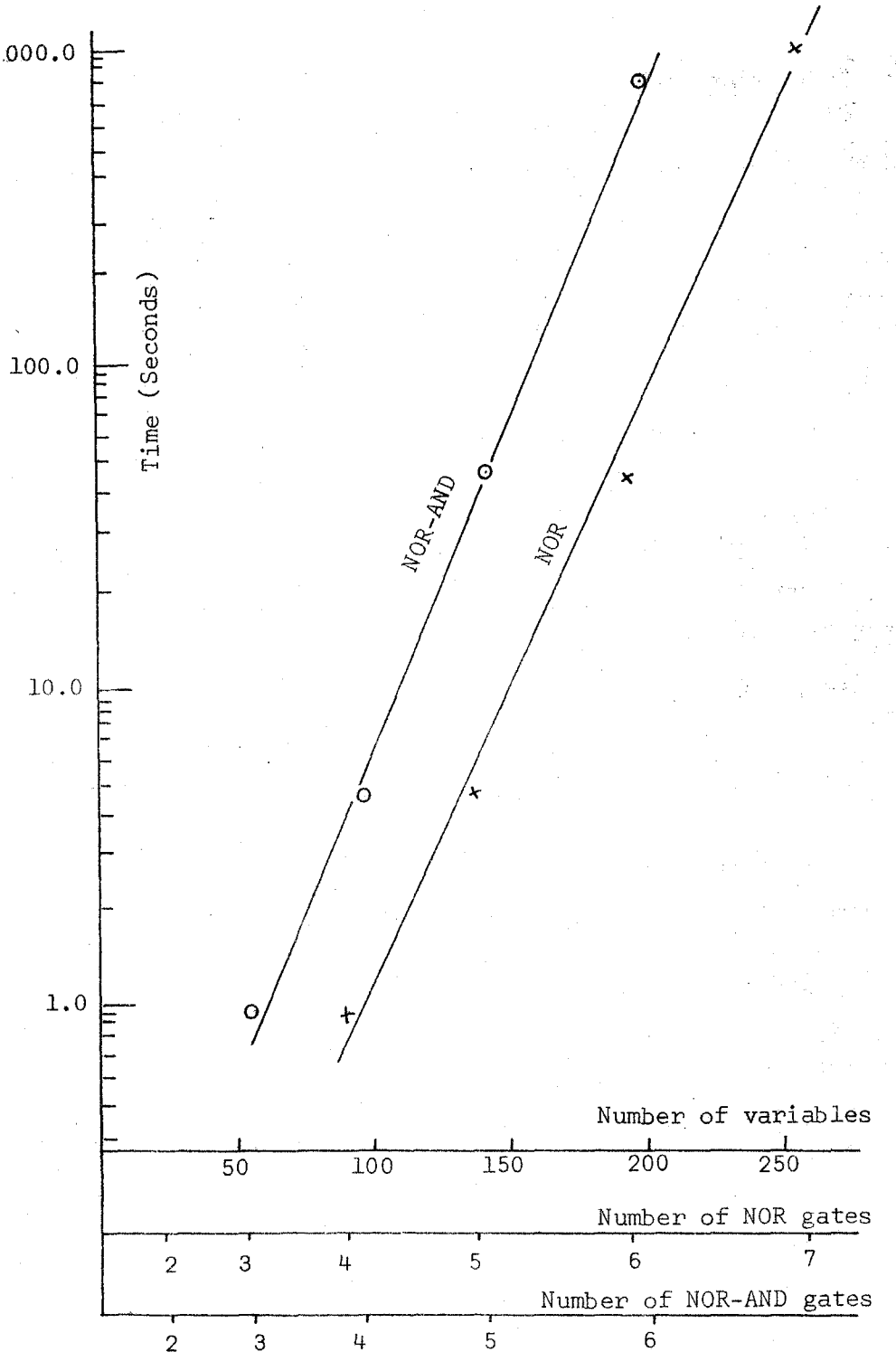


Fig.6.12. Average computation time / function for NOR and NOR-AND synthesis.

6.8 Conclusion

Two logical design problems, one with NOR gates only and the other with NOR-AND combination, formulated as integer linear programs were solved by using the implicit enumeration algorithm. NOR gate network synthesis is important from an engineering view point and has attracted much attention. Integer programming approach to this problem is proved to be computationally feasible and is faster than Hellerman's exhaustive method. Advantages of integer programming approach include its versatility and simplicity to handle a variety of gate types, a variety of network restrictions such as maximum fan-ins restriction and maximum fan-outs restriction, and also various objectives, without changing the algorithm.

Simply changing part of the algorithm i.e., the AGMT-VAR, we can have a program of high speed at the sacrifice of the ease of programming, a program of simplicity and generality at the sacrifice of speed, and a wide range of programs between these two extremes.

Due to the principle of the implicit enumeration algorithm, we can modify and augment the algorithm so that the intrinsic properties of the problem can be fully used. Section 6.5 explored this possibility along with Davidson's work and obtained a improved result in NOR gate network synthesis. Furthermore three networks for the odd parity function were proven to be optimal.

By simply changing the inequalities, different problems can be solved. NOR-AND gate networks were

thus synthesized by using different set of inequalities. Also we believe that this is the first tabulation of all the optimal networks with NOR and AND gates for all three variable functions.

Syntheses of networks with other types of gates such as NOR-NAND combination, and AND-OR with complemented variables are in progress. Also, the further improvement of the efficiency of the algorithm is being attempted.

Chapter 7 Concluding Remarks

With the advent of the recent advances in integrated circuit, large scale integration and others, numerous important changes have been induced in the field of switching theory. New types of gates became available, and new network restrictions were introduced to guarantee a reliable operation. Also the cost criterion involved in the design process has deviated from that used in the classical theory. The situation is much more complicated.

The classical design theories for networks consisting of logical gates, which might be represented by the Quine-McCluskey theory of AND/OR network minimization, seem incapable of accepting such changes.

As stated in Section 1, this thesis is devoted to develop synthesis algorithms which are applicable to those new situations. The thesis includes algorithms designing

(1) threshold gate networks,

(2) negative gate networks,

(3) networks consisting of NOR gates, or other fixed type gates, or their mixture.

Cases (1) (2) are difficult because the designer has to specify the functional form of each gate in the network within the limit of threshold function or negative function. Case (3) is of great practical importance and also difficult because network restrictions are usually imposed, resulting in the networks with more than 3 levels, and a variety of optimality criteria must be adopted.

All the algorithms developed in the thesis are quite different from the classical theory. With these algorithms, threshold gate networks can be designed by hand for functions of about up to 8~9 variables. All the resulting networks appear considerably good though the optimality is not guaranteed. Negative gate networks will be processed by the algorithm in Section 4, which will guarantee the optimality associated with the networks, for functions of up to 8~9 variables or more with the aid of computer. All the optimal networks of NOR gates and NOR/AND gates for functions of up to 3 variables were exhausted as examples of case (3). An advantage of integer programming approach developed for case (3) is that it permits a wide variety of network restrictions and optimality criteria.

Roughly speaking, these algorithms can synthesize networks consisting of at most about 10 gates, considering that only limited computation time is available for designers. This might be regarded as an extremely small number of gates since a practical digital machine usually contains thousands or more number of gates. In fact, that observation would be true in the sense that an ultimate goal is the automation of entire design processes arisen in constructing a digital machine. However, there seems no hope that such algorithm will emerge in the near future. Present trend is rather directed to develop an efficient algorithm of computer aided design, which will produce a reasonably good machine with less effort. In this type of algorithm,

those methods proposed in this thesis may be included as part of the entire procedure, and may prove useful.

Probably, apart from the total design automation, our future effort should be directed to increase the size of networks which permit the application of algorithms, and to take into consideration the other factors such as geometrical quantities. For example, the length of wires in a circuit, their relative position, crossovers and so on, are important factors in actually realizing a reliable machine. Algorithms, which handle these factors as well as others and provide not only an optimal connections of gates but also their placement, would be desirable.

Acknowledgement

The author would like to appreciate Prof. H. Mine of Kyoto University for supervising this thesis. His constant encouragement and invaluable comments on the thesis have greatly helped to complete the work.

The author also wishes to express his thanks to Prof. K. Maeda of Kyoto University for his generous support of the work since the author was a student in his laboratory.

Concerning the topics in Chapter 2 and Chapter 3, the author is indebted to Associate Prof. S. Yajima of Kyoto University, who introduced him to the threshold logic and has always given eager discussions and comments. He also critically read the manuscript of the thesis and provided accurate comments which are sincerely appreciated.

The work in Chapters 4, 5, and 6 was done while the author was a research associate of University of Illinois. The author is grateful to Prof. S. Muroga of University of Illinois for his persistent guidance and support of the work given in this period. The theory in Chapter 4 started from his suggestion and the work in Chapters 5 and 6 was really a continuation of the formulation which he had initiated. Prof. S. Muroga also, as an originator of threshold logic, discussed the topics in Chapters 2 and 3, and it contributed to improve the presentation.

The author has to mention and gratefully acknowledge the comments, suggestions and cooperation offered by all his colleagues, while he was with the Department of Electronics, Kyoto University, while with the Department of Computer Science, University of Illinois, and while with the Department of Applied Mathematics and Physics, Kyoto University. Especially, Mr. Y. Kambayashi, Mr. I. Kawano, Mr. C. R.Baugh and Mr. T. K. Liu have partly participated in completing the theories in Chapters 2 and 3, and in performing the computational experiments in Chapter 6.

References

[1] S. B. Akers, "A truth table method for the synthesis of combinational logic," IRE Transactions on Electronic Computers, vol. EC-10, no. 4, pp. 604-615, December 1961.

[2] _____, "Synthesis of combinational logic using three-input majority gates," Proceedings of the third annual symposium on switching circuit theory and logical design, pp. 150-157, October 1962.

[3] T. Arimoto, "Periodical sequences realizable by an autonomous net of threshold elements" (in Japanese), papers of the Committee on Automaton and Automatic Control, Inst. Elec. Commun. of Japan, March 1963.

[4] E. Balas, "An additive algorithm for solving linear programming with zero-one variables," Operations Research, vol. 13, no. 4, pp. 517-544, July-August 1965.

[5] M. L. Balinski, "Integer programming: methods, uses, computation," Management Science, vol. 12, no. 3, pp. 253-313, November 1965.

[6] C. R. Baugh, T. Ibaraki and S. Muroga, "Computational experience in all-integer, binary variable, integer programming problem using Gomory's all-integer algorithm," Report No. 259, Department of Computer Science, University of Illinois, April 1968.

[7] _____, _____, T. K. Liu and S. Muroga, "Optimum network design using NOR and NOR-AND gates by integer programming," Report No. 293, Department of Computer Science, University of Illinois, January 1969.

[8] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: a survey," Operations Research, vol. 16, no. 3, pp. 548-558, May-June, 1968.

[9] M. A. Breuer, "Implementation of threshold nets by integer linear programming," IEEE Trans. Electronic Computers (Short Notes), vol. EC-14, pp. 950-952, December 1965.

[10] S. H. Cameron, "An estimate of the complexity requisite in a universal decision network," Bionics Symposium,WADD Report 60-600, pp. 197-212, December 1960.

[11] _____, "The generation of minimal threshold nets by an integer program," IEEE Trans. Electronic Computers (Correspondence), vol. EC-13, pp. 299-302, June 1964.

[12] F. T. Chen, "Code for integer linear programming problem for Gomory's algorithm II," Department of Computer Science, University of Illinois, 1968.

[13] C. K. Chow, "Boolean functions realizable with single threshold devices," Proc. IRE (Correspondence) vol. 49, pp. 370-371, January 1961.

[14] _____, "On the characterization of threshold functions," AIEE general fall meeting on SCTLD, September 1961.

[15] C. L. Coates and P. M. Lewis II, "Linearly separable switching functions," Journal of Franklin Institute, vol. 272, pp. 360-410, November 1961.

[16] A. Cobhan, R. Fridshal and J. H. North, "An application of linear programming to the minimization of Boolean functions," Proceedings of the second annual symposium on switching circuit theory and logical design, pp. 3-9, October 1961.

[17] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," IEEE Trans., EC-14, 3, p. 326, June 1965.

[18] G. B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.

[19] E. S. Davidson, "An algorithm for NAND decomposition of combinational switching function," Ph. D. dissertation, Department of Electrical Engineering and Coordinated Science Laboratory, University of Illinois, 1968.

[20] M. L. Dertouzos, Threshold logic, M. I. T. Press 1965.

[21] N. J. Driebeek, "An algorithm for the solution of mixed integer programming problems," Management Science, vol. 12, no. 7, pp. 576-587, March 1966.

[22] C. C. Elgot, "Truth functions realizable by single threshold organs," Switching Circuit Theory and Logical Design, AIEE Special Publication S-134, pp. 225-245, September 1961.

[23] P. Ercoli and L. Mercurio, "Threshold logic with one or more than one threshold," 1962 Proc. IFIP Cong. Amsterdam: North-Holland Publishing Co., pp. 741-746, 1963.

[24] B. Fleischmann, "Computational experience with the algorithm of Balas," Operations Research, vol. 14, no. 1, pp. 153-155, January-February, 1966.

[25] R. J. Freeman, "Computational experience with a 'Balasian' integer programming algorithm," Operations Research, vol. 14, no. 5, pp. 935-942, September-October, 1966.

[26] K. Fukunaga and T. Itoh, "A design theory of recognition functions in self-organizing systems," (in Japanese) J. of Inst. of Elec. and Commun. Eng. of Japan, vol. 46, no. 11, pp. 1672-1679, November 1963.

[27] I. J. Gabelman, "The functional behavior of majority (threshold) elements," Ph. D. dissertation, Dept. of Elec. Engrg., Syracuse University, Syracuse, N. Y., 1961; also "Properties and transformation of single threshold element functions," IEEE Trans. Electronic Computers, vol. EC-13. pp. 680-684, December 1964.

[28] A. M. Geoffrion, "Integer programming by implicit enumeration and Balas' method," SIAM Review, vol. 9, no. 2, pp. 178-190, April, 1967.

[29] _____, "An improved implicit enumeration approach for integer programming," The RAND Corporation, Memorandum RM-5644-PR, June 1968.

[30] E. N. Gilbert, "Lattice theoretic properties of frontal switching functions," Journal of Mathematics and Physics, vol. 33, pp. 57-67, April 1954.

[31] J. F. Gimpel, "The minimization of TANT networks," IEEE TEC, vol. EC-16, no. 1, pp. 18-38, February 1967.

[32] S. Ginsburg, An Introduction to Mathematical Machine Theory. Reading, Mass., Addison-Wesley, 1962.

[33] F. Glover, "A multi phase-dual algorithm for the zero-one integer programming problem," Operations Research, vol. 13, no. 6, pp. 879-919, November-December 1965.

[34] A. J. Goldman, and A. W. Tucker, "Theory of linear programming," in Linear Inequalities and Related Systems, edited by H. W. Kuhn and A. W. Tucker, Princeton University Press, pp. 53-97, 1956.

[35] R. E. Gomory, "An algorithm for the mixed integer problem," The Rand Corporation, P-1885, June 23 1960.

[36] _____, "An all-integer integer programming algorithm," Industrial Scheduling edited by J. R. Muth and G. L. Thompson, Prentice-Hall, 1963.

[37] E. Goto and H. Takahashi, "Some theorems useful in threshold logic for enumerating Boolean functions," 1962 Proc. IFIP Cong. Amsterdam: North-Holland Publishing Co., pp. 747-752, 1963.

[38] C. H. Gustafson, D. R. Haring, A. K. Susskind, and T. G. Willes-Sanford, "Synthesis of counters with threshold elements," IEEE Conf. Rec., 6th Ann. Symp. on Switching Circuit Theory and Logical Design (Ann Arbor, Mich., October 6-8, 1965), pp. 25-35.

[39] G. Hadley, Linear programming, Addison-Wesley, Addison-Wesley Series in Industrial Management, 1962.

[40] F. O. Hadlock and C. L. Coates, "Realization of sequential machines with threshold elements," IEEE Conf. Rec., 7th Ann. Symp. on Switching and Automata Theory (Berkeley, Calif., October 26-28, 1966), pp. 162-183.

[41] J. Haldi, "25 integer programming test problems," Working Paper no. 43, Graduate School of Business, Stanford University, December 1964.

[42] _____ and L. M. Isaacson, "A computer code for integer solutions to linear programs," Operations Research, vol. 13, no. 6, pp. 946-959, November-December 1965.

[43] D. R. Haring, "Multi-threshold threshold elements," IEEE Trans. on Electronic Computers, vol. EC-15, no. 1, pp. 45-65, February 1966.

[44] L.Hellerman,"A catalog of three variable OR-invert and AND-invert logical circuits," IEEE Trans. on Electronic Computers, vol. EC-12, no. 3, pp. 198-223, June 1963.

[45] J. E. Hopcroft and R. L. Mattson, "Synthesis of minimal threshold logic networks," IEEE Trans. on Electronic Computers, vol. EC-14, pp. 552-560, August 1965.

[46] T. Ibaraki, "Theory of threshold functions," Master thesis, Department of Electronics, Faculty of Engineering, Kyoto University, March 1965.

[47] _____, and S. Muroga, "Adaptive linear classifier by linear programming," Report no. 284, Department of Computer Science, University of Illinois, September 1968*

[48] _____, T. K. Liu, C. R. Baugh and S. Muroga, "An implicit enumeration program for zero-one integer programming," Report no. 305, Department of Computer Science, University of Illinois, January 1969.

[49] _____, and S. Muroga, "Implicit enumeration algorithm of integer programming on Illiac IV," Report no. 319, Department of Computer Science, University of Illinois, January 1969.

[50] _____ and _____, "Minimization of switching networks using negative functions," Report no. 309, Department of Computer Science, University of Illinois, February 1969.

[51] K. Ibuki, K. Naemura and A. Nozaki, "General theory of complete sets of logical functions," (in Japanese) J. of Inst. of Elec. Commun. Eng. of Japan, vol. 46, no. 7, pp. 934-940, July 1963.

*To be published in the January, 1970, issue of IEEE Trans.on Cybernetics and System Science.

[52] N. Ikeno, A. Hashimoto, and K. Naito, "A table of four-variable minimal NAND circuits," (in Japanese) Electrical Communication Laboratory Technical Journal, Extra issue No. 26, Nippon Telegraph and Telephone Public Corporation, 1968.

[53] N. Ishii, M. Kimura, and N. Honda, "Modes of periodical sequences by nets of threshold elements," (in Japanese), papers of the Committee on Automaton and Automatic Control, Inst. Elec. Commun. of Japan, July 1965.

[54] _____ and _____, "Testing and synthesis of threshold networks realizing a given state transition diagram," (in Japanese) J. of Inst. of Elec. and Commun. Eng. of Japan, vol. 51-C, no. 6, pp. 259-266, June 1968.

[55] _____ and _____, "Generation of threshold functions based on the tables of connecting edges," (in Japanese) J. of Inst. of Elec. and Commun. Eng. of Japan, vol. 51-C, no. 10, pp. 447-454, October 1968.

[56] P. Kaszerman, "A geometric test-synthesis procedure for a threshold device," Information and Control, vol. 6, pp. 381-398, December 1963.

[57] W. H. Kautz, "The realization of symmetric switching functions with linear-input logical elements," IRE Trans. Electronic Computers, vol. EC-10, pp. 371-378, September 1961.

[58] K. Krohn and J. Rhodes, "Nets of threshold elements," Information and Control, vol. 8, pp. 579-588, 1965.

[59] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems, Econometrica, vol. 28, no. 3, pp. 497-520, July 1960.

[60] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: a survey," Operations Research, vol. 14, no. 4, pp. 699-719, July-August 1966.

[61] C. E. Lemke and K. Spielberg, "Discrete search algorithm for zero-one and mixed-integer programming," Operations Research, vol. 15, no. 5, pp. 892-914, September-October 1967.

[62] P. M. Lewis II and C. L. Coates, Threshold logic, Wiley, N. Y., 1967.

[63] T. K. Liu, "A code for zero-one integer linear programming by implicit enumeration," Master Thesis, Department of Computer Science, University of Illinois, 1968.

[64] O. L. Mangasarian, "Linear and nonlinear separation of patterns by linear programming," Operations Research, vol. 13, no. 3, pp. 442-452, May-June 1965.

[65] G. T. Martin, "Directed network models for discrete programming," Paper presented at the International Symposium on Mathematical Programming, London School of Economics, July 1964.

[66] E. J. McCluskey Jr., "Minimization of Boolean functions," Bell Sys. Tech. J., vol. 35, pt. 2, pp. 1417-1444, November 1956.

[67] _____, Introduction to the theory of switching circuits, McGraw-Hill Inc., New York, 1965.

[68] C. E. Miller, A. W. Tucker and R. A. Zemlin, "Integer programming formulation of traveling salesman problems," Journal of ACM, vol. 7, pp. 326-329, 1960.

[69] R. C. Minnick, "Linear-input logic," IRE Trans. Electronic Computers, vol. EC-10, pp. 6-16, March 1961.

[70] S. Muroga, "Logical elements on majority decision principle and complexity of their circuits," International Conference on Information Processing, UNESCO, June 1959, Published by Columbia University Press (Japanese version was delivered at Fukuoka, May 1958).

[71] _____, I. Toda, "Theory of majority decision elements," J. of Inst. of Elec. Commun. Eng. of Japan, vol. 43, no. 10, pp. 1071- , October 1960.

[72] _____, _____, and S. Takasu, "Theory of majority decision elements," J. Franklin Inst., vol. 271, pp. 376-418, May 1961.

[73] _____, "Majority logic and problems of probabilistic behavior," Self-Organizing Systems, published by Spartan Books, pp. 243-281, 1962.

[74] _____, "Functional forms of dual-comparable functions and a necessary and sufficient condition for realizability of a majority function," IEEE Trans. Communication and Electronics, vol. 83, pp. 474-486, September 1964.

[75] _____, "Lower bounds of the number of threshold functions and a maximum weight," IEEE Trans. on Electronic Computers, vol. EC-14, pp. 136-148, April 1965 (originally AIEE 3rd Annual Symp. on Switching Circuit Theory and Logical Design, Chicago, Ill., 1962).

[76] _____, "Threshold logic," Department of Computer Science, University of Illinois, Urbana, Ill., lecture notes for EE497 and EE498, 1965-1966.

[77] _____, "Logical design of an optimum network by integer linear programming," Lecture Notes of Threshold Logic, 1965-1966 School Year. Also File No. 700, Digital Computer Laboratory, University of Illinois, July 11, 1966.

[78] S. Muroga and I. Toda, "Lower bound of the number of threshold functions," IEEE Trans. Electronic Computers (Short Notes), vol. EC-15, pp. 805-806, October 1966.

[79] _____, T. Tsuboi and C. R. Baugh, "Enumeration of threshold functions of eight variables," Department of Computer Science, University of Illinois, Report No. 245, August 1967.

[80] _____ and T. Ibaraki, "Logical design of an optimum network by integer linear programming-part I," Report no. 264, Department of Computer Science, University of Illinois, July 1968.

[81] _____ and _____, "Logical design of an optimum network by integer linear programming-part II," Report no. 289, Department of Computer Science, University of Illinois, December 1968.

[82] N. J. Nilsson, Learning machines, New York: McGraw-Hill, 1963.

[83] T. Ozawa, "On multi-threshold threshold elements." (in Japanese), paper of the committee on Automaton and Automatic Control, Institute of Electrical Communication Engineers of Japan, February 1964.

[84] M. C. Paull and E. J. McCluskey, Jr., "Boolean functions realizable with single threshold devices," Proc. IRE (Correspondence), vol. 48, pp. 1335-1337, July 1960.

[85] C. C. Peterson, "Computational experience with variants of the Balas algorithm applied to the selection of R and D projects," Management Science, vol. 13, no. 9, pp. 736-750, May 1967.

[86] E. L. Post, The two-valued iterative systems of mathematical logic, Annals of mathematical studies, no. 5, Princeton University Press, Princeton, 1941.

[87] W. V. Quine, "The problem of simplifying truth functions," Am. Math. Monthly, vol. 59, pp. 521-531, October 1952.

[88] _____, "Two theorems about truth functions," Bol. Soc. Math. Mexicana, vol. 10, pp. 64-70, 1953.

[89] F. Rosenblatt, Principle of Neurodynamics, Spartan Books, Washington D. C., 1961.

[90] T. Sakai, H. Fukutome, and I. Sugata, "Analysis of nets of threshold elements" (in Japanese), papers of the Committee on Automaton and Automatic Control, Inst. Elec. Commun. of Japan, November 1965.

[91] _____, and I. Sugata, "An analysis of threshold element network," (in Japanese) J. of Inst. of Elec. and Commun. Eng. of Japan, vol. 51-C, n0. 5, pp. 175-182, May 1968.

[92] D. R. Smith, "Bounds on the number of threshold functions," IEEE Trans. Electronic Computers (Short Notes), vol. EC-15, pp. 368-369, June 1966.

[93] F. W. Smith, "Pattern classifier design by linear programming," IEEE Trans. on Computers, vol. C-17, no. 4, pp. 367-372, April 1968.

[94] E. P. Stabler, "Threshold gate network synthesis," IEEE Conf. Rec., 6th Ann. Symp. on Switching Theory and Logical Design (Ann Arbor, Mich., October 6-8, 1965), pp. 5-11.

[95] I. Sugata, "A synthesis method of threshold element network," (in Japanese), J. of Inst. of Elec. and Commun. Eng. of Japan, vol. 51-C, no. 12, pp. 581-588, December 1968.

[96] K. Taniguchi, N. Tokura, T. Kasami and H. Ozaki, "Logical functions realizable by a planar NAND network," J. of Inst. of Elec. Commun. Eng. of Japan, vol. 51-C, no. 2, pp. 59-65, February 1968.

[97] E. Thomas and J. D. Callan, "Slow, but small, may win the race," Electronics, vol. 40, no. 4, pp. 179-182, February 20, 1967.

[98] R. O. Winder, "Threshold logic," Ph. D. dissertation, Dept. of Math., Princeton University, Princeton, N. J., 1962; also "Single stage threshold logic," Switching Circuit Theory and Logical Design, AIEE Special Publication S-134, pp. 55-64, September 1961.

[99] _____, "Bounds on threshold gate realizability," IEEE Trans. Electronic Computers, vol. EC-12, no. 5, p. 561, October 1963.

[100] _____, "Enumeration of seven-argument threshold functions," IEEE Trans. Electronic Computers, vol. EC-14, pp. 315-325, June 1965.

[101] _____, "The status of threshold logic," First annual Princeton Conference on Information Sciences and Systems, March 1967.

[102] S. Yajima and T. Ibaraki, "On the lower bound of the number of threshold functions," (in Japanese), J. of Inst. of Elec. and Commun. Eng. of Japan, vol. 48, no. 3, pp. 439-440, March, 1965.

[103] _____ and _____, "A lower bound of the number of threshold functions," IEEE Trans. Electronic Computers, vol. EC-14, no. 6, p. 926, December 1965.

[104] _____ and _____, "Successive approximation realization of a threshold function by its characteristic vector," (in Japanese), papers of the Committee on Automaton and Automatic Control, Inst. Elec. Commun. of Japan. January 1967.

[105] _____ and _____, "On relation between a logic function and its characteristic vector," (in Japanese) J. of Inst. of Elec. and Commun. Eng. of Japan, vol. 50, no. 3, pp. 25-32, March 1967.

[106] S. Yajima and T. Ibaraki, "A theory of completely monotonic functions and its applications to threshold logic," (in Japanese) J. of Inst. of Elec. and Commun. Eng. of Japan, vol. 50. no. 10, pp. 192-199, October 1967.

[107] _____ and _____, "Realization of arbitrary logic functions by completely monotonic functions and its applications to threshold logic," (in Japanese) J. of Inst. of Elec. and Commun. Eng. of Japan, vol. 50, no. 10, pp. 200-207, October 1967.

[108] _____ and _____, "A theory of completely monotonic functions and its applications to threshold logic," IEEE Trans. Electronic Computers, vol. EC-17, pp. 214-229, March 1968.

[109] _____ and _____, "Realization of arbitrary logic functions by completely monotonic functions and its applications to threshold logic," IEEE Trans. Computers, vol. C-17, pp. 338-351, April 1968.

[110] _____, _____, and I. Kawano, "On autonomous logic nets of threshold elements," IEEE Trans. Computers, vol. C-17, no. 4, pp. 385-391, April 1968.

[111] Y. T. Yen, "A mathematical model characterizing four-phase MOS circuits for logical simulation," IEEE Transactions on Computers, vol. C-17, no. 9, pp. 822-826, September 1968.

Supplement

[112] M. Bloch and J. Moravek, "Bounds of the number of threshold functions," Information processing machines, no. 13, pp. 67-73, 1967.

[113] M. A. Breuer, "General survey of design automation of digital computers," Proc. of the IEEE, vol.54, no.12, pp. 1708-1721, December 1966.

[114] E. Goto, "Threshold, majority and bilateral switching devices," Symposium on the application of switching theory in space technology, Stanford University Press., pp. 47-67, 1963.

[115] M. Ito, Y. Inagaki and M. Fukumura, "Synthesis of NAND networks with fan-ins and fan-outs restrictions," (in Japanese) Paper of the Commitee on Automaton and Information Theory, Institute of Electrical Communication Engineers of Japan, September, 1967.

[116] H. Nomura, T.Kitahashi, K. Tezuka and Y. Kasahara, "On the multi-threshold threshold functions and their realization (1)," (in Japanese) Paper of Commitee on Automaton, Institute of Electronics and Communication Engineers of Japan, June 1968.

[117] I. Toda and K. Naito, "Synthesis method of NAND networks by integer programming," (in Japanese) Paper of Commitee on Computers, Institute of Electronics and Communication Engineers of Japan, December 1966.

[118] T. Tsuboi, " A logical design of circuits representing Boolean functions with four or less variables by means of three-input Parametrons and four-input Parametrons," (in Japanese) Information Processing in Japan, vol.4, pp. 20-40, 1964.

Appendix. Tabulation of Optimal NOR-AND Networks

Here listed are all the optimal networks for each of all functions of up through three variables, using NOR and/or AND gates.

The networks which have the minimum number of interconnections and connections among the networks with the minimum number of gates are chosen as optimal network.

The procedure for obtaining the optimal network diagram for a given function follows that of Hellerman's [8]. A function can be represented by a truth table as shown below, by specifying the values of f_1, f_2, ..., f_8 where f_1, ..., f_8 show the values of the given f for input vectors in the same rows. a, b and c denote the variables of f.

	a	b	c
f_1	0	0	0
f_2	0	0	1
f_3	0	1	0
f_4	0	1	1
f_5	1	0	0
f_6	1	0	1
f_7	1	1	0
f_8	1	1	1

Let us write eight binary numbers $f_1, \ldots, f_8$ as follows.

$$\underbrace{f_8 \quad f_7}_{0_2} \qquad \underbrace{f_6 \quad f_5 \quad f_4}_{0_1} \qquad \underbrace{f_3 \quad f_2 \quad f_1}_{0_0}$$

grouping the f_i's as shown, we obtain three octal number $0_2\, 0_1\, 0_0$. This octal number is used throughout Appendix to identify a function.

In Table A2 and A3, only representatives of equivalence class obtained by permutation of variables are listed, reducing 256 functions into 80 representatives. The representative of a given function f can be easily derived by using Table A1, which is taken from Hellerman [44]. The procedure is illustrated by an example.

Suppose that the function f has the number 321. The entry for this number in Table A1 is 5*213. This means that f is equivalent to function 213 by applying permutation 5 which is (ac) as also shown in Table A1. Table A2 shows that function 213 has optimal networks with network number 49 and 56. Then Table A3 gives us the actual optimal networks of function 213. To obtain optimal networks of 321, we apply the inverse of permutation 5, which is also (ac), to these networks in Table A3.

Twelve of the 80 three-variable functions listed are degenerate in the sense that they are independent of at least one variable. In Table A1, those degenerate functions appear with minus signs. The network diagram numbers for the degenerate functions have the suffix D.

They are grouped together at the beginning of Table A3.

Table A3 shows all the optimal networks obtained for each function without imposing fan-in restrictions. However all the networks except that of function 026 are optimal even if the fan-in restriction that each gate can have at most three input interconnections is imposed. Function 026 needs one more gate if the above fan-in restriction is imposed. Optimal networks in this case are shown in Fig. A1.

Note that if a given function is symmetric in some variables, say a and b, then, among all the optimal networks obtained by permuting these symmetric variables, a and b, only one network is listed in Table A2 and Table A3. The rest of networks can be obtained by simply exchanging the connection from the external variables according to the permutation of variables.

	0	1	2	3	4	5	6	7
	-1*000	1*001	1*002	-1*003	3*002	-3*003	1*006	1*007
	1*010	1*011	-1*012	1*013	-4*012	4*013	1*016	-1*017
	2*002	-2*003	2*006	2*007	3*006	3*007	1*026	1*027
30	1*030	1*031	1*032	1*033	4*032	4*033	1*036	1*037
40	2*010	2*011	-6*012	6*013	2*030	2*031	6*032	6*033
50	1*050	1*051	1*052	1*053	1*054	1*055	1*056	1*057
60	-2*012	2*013	2*016	-2*017	2*032	2*033	2*036	2*037
70	6*054	6*055	6*056	6*057	-1*074	1*075	1*076	-1*077
100	3*010	3*011	3*030	3*031	-3*012	3*013	3*032	3*033
110	3*050	3*051	4*054	4*055	3*052	3*053	4*056	4*057
120	-5*012	5*013	5*032	5*033	3*016	-3*017	3*036	3*037
130	3*054	3*055	-3*074	3*075	3*056	3*057	3*076	-3*077
140	2*050	2*051	2*054	2*055	5*054	5*055	-2*074	2*075
150	1*150	1*151	1*152	1*153	3*152	3*153	1*156	1*157
160	2*052	2*053	2*056	2*057	5*056	5*057	2*076	-2*077
170	2*152	2*153	2*156	2*157	3*156	3*157	1*176	1*177
200	1*200	1*201	1*202	1*203	3*202	3*203	1*206	1*207
210	-1*210	1*211	1*212	1*213	4*212	4*213	1*216	1*217
220	2*202	2*203	2*206	2*207	3*206	3*207	1*226	1*227
230	1*230	-1*231	1*232	1*233	4*232	4*233	1*236	1*237
240	-2*210	2*211	6*212	6*213	2*230	-2*231	6*232	6*233
250	1*250	1*251	-1*252	1*253	1*254	1*255	1*256	-1*257
260	2*212	2*213	2*216	2*217	2*232	2*233	2*236	2*237
270	6*254	6*255	6*256	-6*257	1*274	1*275	1*276	1*277
300	-3*210	3*211	3*230	-3*231	3*212	3*213	3*232	3*233
310	3*250	3*251	4*254	4*255	-3*252	3*253	4*256	-4*257
320	5*212	5*213	5*232	5*233	3*216	3*217	3*236	3*237
330	3*254	3*255	3*274	3*275	3*256	-3*257	3*276	3*277
340	2*250	2*251	2*254	2*255	5*254	5*255	2*274	2*275
350	1*350	1*351	1*352	1*353	3*352	3*353	-1*356	1*357
360	-2*252	2*253	2*256	-2*257	5*256	-5*257	2*276	2*277
370	2*352	2*353	-2*356	2*357	-3*356	3*357	1*376	-1*377

Explanatory Example
Class of 321 is given by word at intersection of row 320 and column 1, 5*213.
This says 321 is in class of 213 by permutation 5.
Negative permutation means the function is degenerate.

Permutation 1 is the identity
Permutation 2 is (abc)
Permutation 3 is (acb)
Permutation 4 is (bc)
Permutation 5 is (ac)
Permutation 6 is (ab)

Table A1. Equivalent classes of functions of three variables.

Table A2. Catalog of all the optimal NOR-AND networks of three variable functions.

Function (octal)	Functional expression	Network number	No. of gates		No. of interconnections and connections	No. of levels
			NOR	AND		
000	0	1D	0	0	0	0
003	a'b'	5D	1	0	2	1
012	a'c	8D	2	0	3	2
		9D	1	1	3	2
017	a'	4D	1	0	1	1
074	ab' + a'b	13D	2	1	6	2
077	a' + b'	10D	1	1	3	2
210	bc	6D	0	1	2	1
231	bc + b'c'	14D	3	1	7	3
		15D	3	1	7	3
252	c	3D	0	0	0	0
257	a' + c	11D	3	0	4	3
		12D	2	1	4	3
356	b + c	7D	2	0	3	2
377	1	2D	0	0	0	0
001	a'b'c'	1	1	0	3	1
002	a'b'c	3	2	0	4	2
		7	1	1	4	2
006	a'b'c + a'bc'	15	2	1	7	2
007	a'b' + a'c'	8	1	1	4	2
010	a'bc	6	1	1	4	2
011	a'bc + a'b'c'	54	3	1	8	3
		61	3	1	8	3
013	a'b' + a'c	18	3	0	5	3

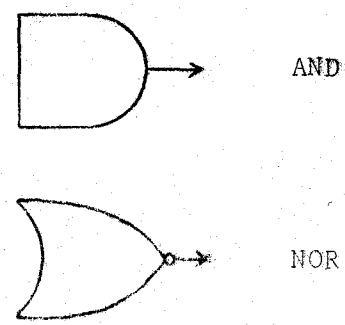
Function (octal)	Functional expression	Network number	No. of gates		No. of interconnections and connections	No. of levels
			NOR	AND		
		22	2	1	5	3
016	a'b + a'c	4	2	0	4	2
026	a'b'c + a'bc' + ab'c'	68	2	3	13	2
027	a'b' + b'c' + a'c'	30	1	3	9	2
030	ab'c' + a'bc	70	4	1	10	3
		71	3	2	10	3
		88	4	1	10	4
031	a'bc + b'c'	85	4	1	9	4
032	a'c + ab'c'	28	2	2	9	2
033	a'c + b'c'	34	3	1	7	3
		44	2	2	7	3
036	a'b + a'c + ab'c'	29	2	2	10	2
037	a' + b'c'	17	1	2	6	2
050	a'bc + ab'c	27	3	1	8	2
		55	2	2	8	3
051	a'b'c' + a'bc + ab'c	79	3	2	11	3
052	a'c + b'c	11	2	1	5	2
		26	1	2	5	3
053	a'b' + a'c + b'c	36	3	1	8	3
054	a'b + ab'c	47	2	2	8	3
055	a'b + a'c' + ab'c	74	3	2	11	3
		78	3	2	11	3
056	a'b + b'c	12	2	1	6	2
057	a' + b'c	33	3	1	7	3
		43	2	2	7	3
075	a'b + ab' + a'c'	35	3	1	8	3
		45	2	2	8	3
076	a'b + ab' + a'c	14	2	1	7	2

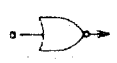
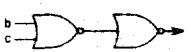
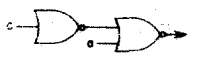
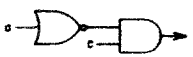
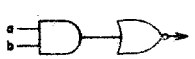
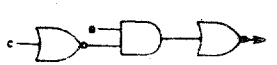
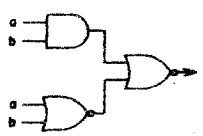
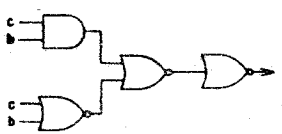
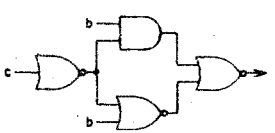
Function (octal)	Functional expression	Network number	No. of gates		No. of interconnections and connections	No. of levels
			NOR	AND		
150	a'bc + ab'c + abc'	80	3	2	11	3
151	a'bc + ab'c + abc' + a'b'c'	101	3	3	14	3
152	a'c + b'c + abc'	62	2	2	8	3
153	a'c + a'b' + b'c + abc'	76	3	2	11	3
156	a'c + b'c + bc'	13	2	1	7	2
157	a' + b'c + bc'	37	3	1	9	3
		46	2	2	9	3
176	ab' + bc' + a'c	16	2	1	8	2
177	a' + b' + c'	9	1	1	4	2
200	abc	2	0	1	3	1
201	abc + a'b'c'	52	3	1	9	3
202	abc + a'b'c	73	4	1	9	3
		77	4	1	9	3
		82	3	2	9	4
		90	3	2	9	4
203	abc + a'b'	50	3	1	8	3
206	a'b'c + a'bc' + abc	100	4	2	12	3
		104	4	2	12	4
		106	4	2	12	4
		109	4	2	12	4
		110	4	2	12	4
		111	4	2	12	4
		112	4	2	12	4
207	a'b' + a'c' + abc	81	3	2	9	3

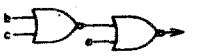
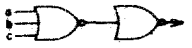
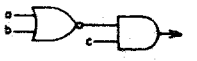
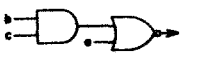
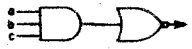
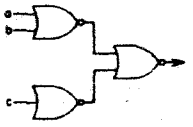
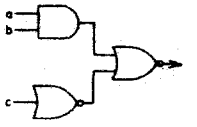
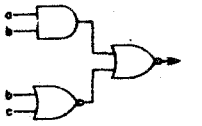
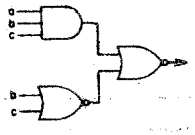
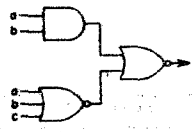
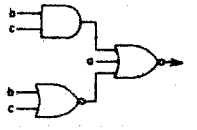
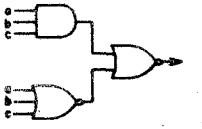
Function (octal)	Functional expression	Network number	No. of gates		No. of interconnections and connections	No. of levels
			NOR	AND		
		89	3	2	9	4
		94	3	2	9	4
211	bc + a'b'c'	51	3	1	8	3
212	a'c + bc	31	4	0	6	3
		38	3	1	6	3
		64	3	1	6	4
		66	2	2	6	4
213	a'b' + bc	49	3	1	7	3
		56	3	1	7	3
216	a'b + a'c + bc	72	4	1	9	3
		86	4	1	9	4
		91	3	2	9	4
		95	4	1	9	4
		96	4	1	9	4
217	a' + bc	48	3	1	6	3
		67	2	2	6	4
226	abc + ab'c' + a'bc' + a'b'c	105	4	2	12	4
227	a'b' + a'c' + b'c' + abc	99	4	2	12	3
		102	4	2	12	4
230	ab'c' + bc	84	4	1	9	4
		87	3	2	9	4
		97	3	2	9	4
232	a'c + bc + ab'c'	92	4	1	9	4
		93	3	2	9	4
233	a'b' + b'c' + bc	57	3	1	8	3

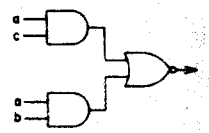
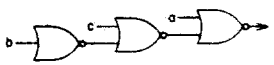
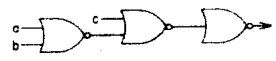
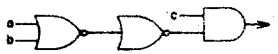
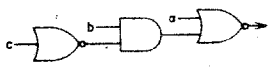
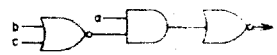
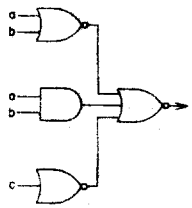
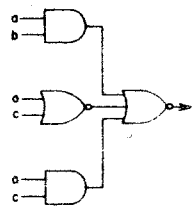
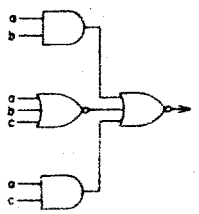
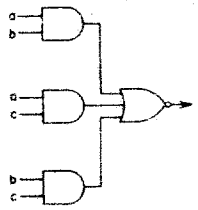
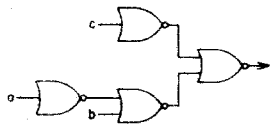
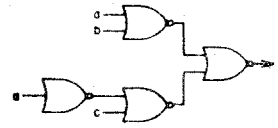
Function (octal)	Functional expression	Network number	No. of gates		No. of interconnections and connections	No. of levels
			NOR	AND		
236	a'c + bc + a'b + ab'c'	98	4	2	12	3
		103	4	2	12	4
		107	4	2	12	4
		108	4	2	12	4
237	a' + bc + b'c'	69	4	1	9	3
		83	3	2	9	4
250	ac + bc	10	3	0	5	2
		21	2	1	5	3
251	ac + bc + a'b'c'	59	3	1	8	3
253	a'b' + c	20	3	0	5	3
254	ac + a'b	32	4	0	7	3
		39	3	1	7	3
255	ac + bc + a'c'	58	3	1	8	3
256	a'b + c	63	4	0	6	4
		65	3	1	6	4
274	ab' + ac + a'b	40	3	1	8	3
275	ab' + ac + a'b + a'c'	60	3	1	9	3
276	a'b + ab' + c	41	3	1	9	3
277	a' + b' + c	23	2	1	5	3
350	ab + ac + bc	42	3	1	8	3
351	ab + ac + bc + a'b'c'	75	3	2	11	3
352	ab + c	25	2	1	5	3
353	ab + a'b' + c	53	3	1	8	3
357	a' + b + c	19	3	0	5	3
		24	2	1	5	3
376	a + b + c	5	2	0	4	2

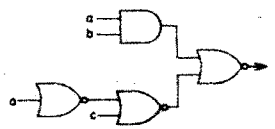
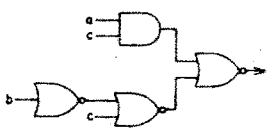
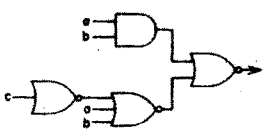
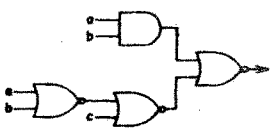
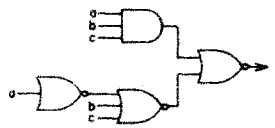
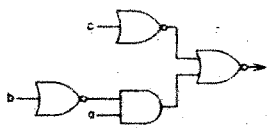
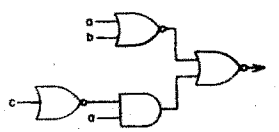
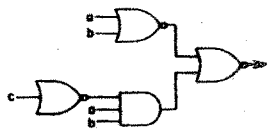
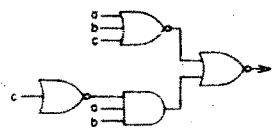
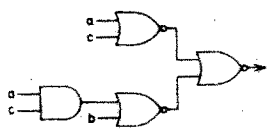
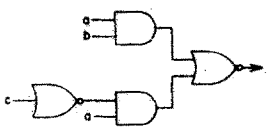
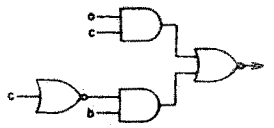
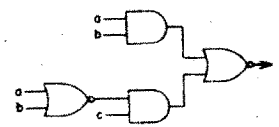
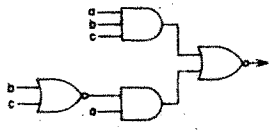
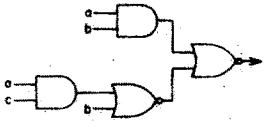
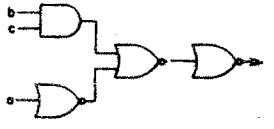
Table A3. List of all the optimal NOR-AND combination networks of three variable functions.



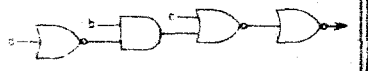
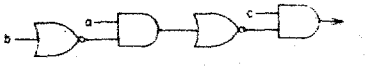
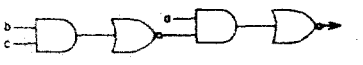
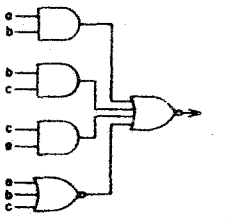
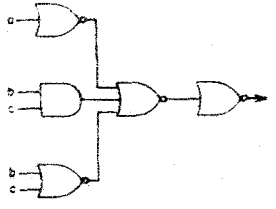
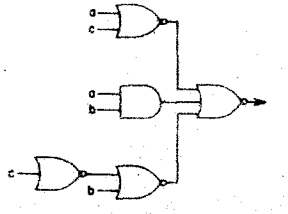
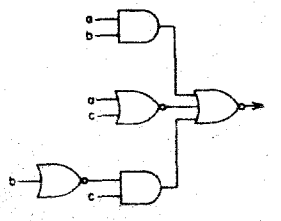
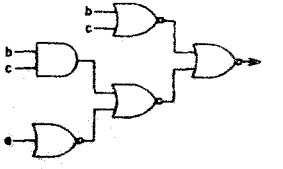
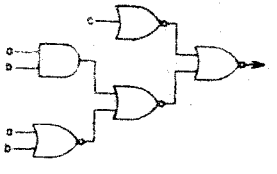
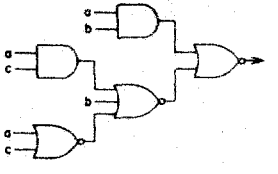
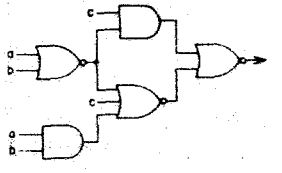
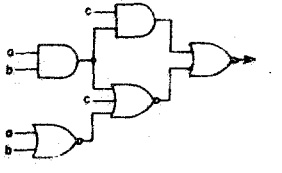
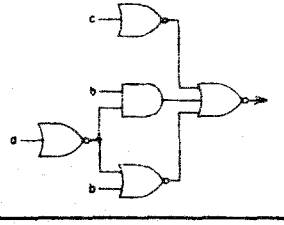
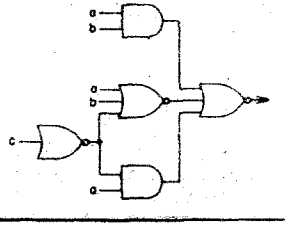
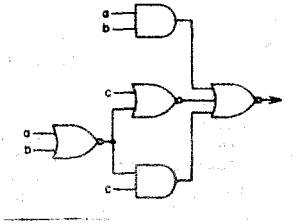
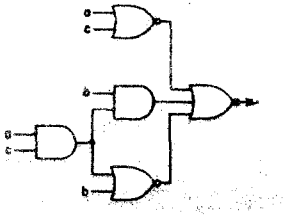
NO	FUNCTION		NO	FUNCTION		NO	FUNCTION		NO	FUNCTION	
1D	000	0	2D	377	1	3D	252	c	4D	017	a'
											
5D	003	a'b'	6D	210	bc	7D	356	b + c	8D	012	a'c
											
9D	012	a'c	10D	077	a'+ b'	11D	257	a'+c	12D	257	a'+c
											
13D	074	ab'+a'b	14D	231	bc+b'c'	15D	231	bc +b'c'			
											

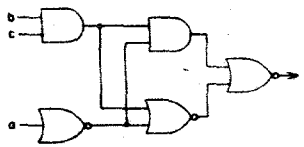
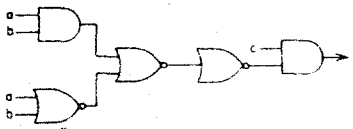
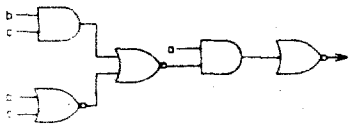
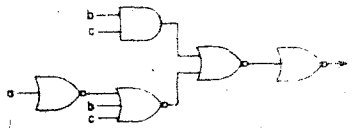
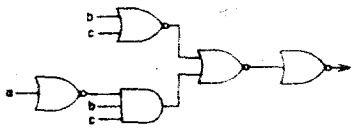
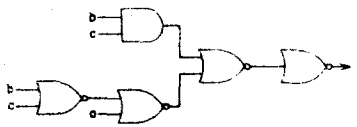
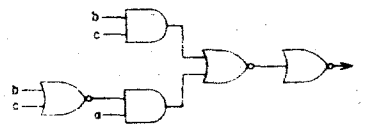
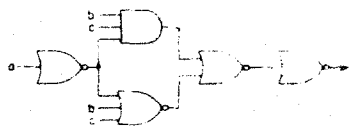
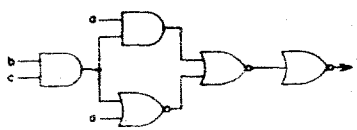
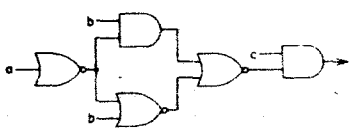
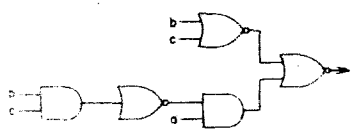
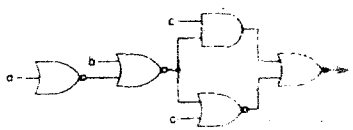
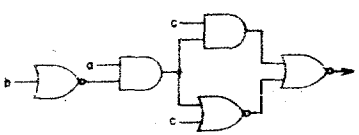
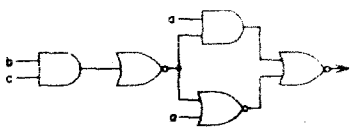
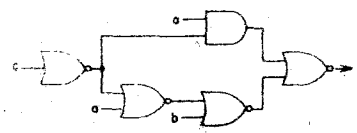
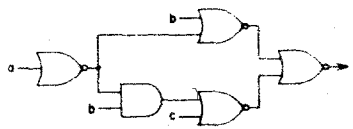
NO	FUNCTION		NO	FUNCTION		NO	FUNCTION		NO	FUNCTION	
1	001	a'b'c'	2	200	abc	3	002	a'b'c	4	016	a'b+a'c
											
5	376	a+b+c	6	010	a'bc	7	002	a'b'c	8	007	a'b'+a'c'
											
9	177	a'+b'+c'	10	250	ac+bc	11	052	a'c+b'c	12	056	a'b+b'c
											
13	156	a'c+b'c+bc'	14	076	a'b+ab'+a'c	15	006	a'b'c+a'bc'	16	176	ab'+bc'+a'c
											

NO	FUNCTION		NO	FUNCTION		NO	FUNCTION		NO	FUNCTION	
17	037	a'+b'c'	18	013	a'b'+a'c	19	357	a'+b+c	20	253	a'b'+c
											
21	250	ac+bc	22	013	a'b'+a'c	23	277	a'+b'+c	24	357	a'+b+c
											
25	352	ab+c	26	052	a'c+b'c	27	050	a'bc+ab'c	28	032	a'c+ab'c'
											
29	036	a'b+a'c+ab'c'	30	027	a'b'+b'c'+a'c'	31	212	a'c+bc	32	254	ac+a'b
											

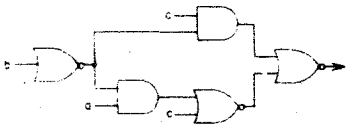
NO		FUNCTION	NO		FUNCTION	NO		FUNCTION	NO		FUNCTION
33	057	a'+b'c	34	033	a'c+b'c'	35	075	a'b+ab'+a'c'	36	053	a'b'+a'c+b'c
											
37	157	a'+b'c+bc'	38	212	a'c+bc	39	254	ac+a'b	40	274	ab'+ac+a'b
											
41	276	a'b+ab'+c	42	350	ab+ac+bc	43	057	a'+b'c	44	033	a'c+b'c'
											
45	075	a'b+ab'+a'c'	46	157	a'+b'c+bc'	47	054	a'b+ab'c	48	217	a'+bc
											

NO		FUNCTION	NO		FUNCTION	NO		FUNCTION	NO		FUNCTION
49	213	a'b'+bc	50	203	abc+a'b'	51	211	bc+a'b'c'	52	201	abc+a'b'c'
53	353	ab+a'b'+c	54	011	a'bc+a'b'c'	55	050	a'bc+ab'c	56	213	a'b'+bc
57	233	a'b'+b'c'+bc	58	255	ac+bc+a'c'	59	251	ac+bc+a'b'c'	60	275	ab'+ac+a'b+a'c'
61	011	a'bc+a'b'c'	62	152	a'c+b'c+abc'	63	256	a'b+c	64	212	a'c+bc

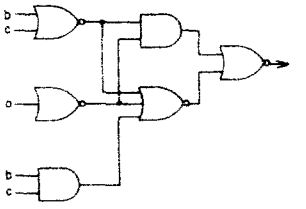
NO		FUNCTION	NO		FUNCTION	NO		FUNCTION	NO		FUNCTION
65	256	a'b + c	66	212	a'c + bc	67	217	a' + bc	68	026	a'b'c + a'bc' + ab'c'
											
69	237	a' + bc + b'c'	70	030	ab'c' + a'bc	71	030	ab'c' + a'bc	72	216	a'b + a'c + bc
											
73	202	abc + a'b'c	74	055	a'b + a'c' + ab'c	75	351	ab + ac + bc + a'b'c'	76	153	a'c + a'b' + b'c + abc'
											
77	202	abc + a'b'c	78	055	a'b + a'c' + ab'c	79	051	a'b'c' + a'bc + ab'c	80	150	a'bc + ab'c + abc'
											

NO		FUNCTION	NO		FUNCTION	NO		FUNCTION	NO		FUNCTION
81	207	a'b'+a'c'+abc	82	202	abc+a'b'c	83	237	a'+bc+b'c'	84	230	ab'c'+bc
											
85	031	ab'c+b'c'	86	216	a'b+a'c+bc	87	230	ab'c'+bc	88	030	ab'c'+a'bc
											
89	207	a'b'+a'c'+abc	90	202	abc+a'b'c	91	216	a'b+a'c+bc	92	232	a'c+bc+ab'c'
											
93	232	a'c+bc+ab'c'	94	207	a'b'+a'c'+abc	95	216	a'b+a'c+bc	96	216	a'b+a'c+bc
											

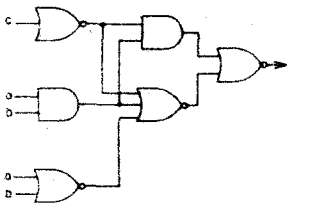
NO		FUNCTION
97	230	a'b'c'+bc



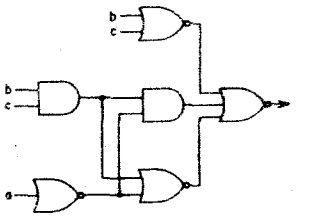
NO		FUNCTION
98	236	a'c + bc + a'b + ab'c'



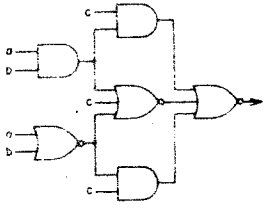
NO		FUNCTION
99	227	a'b' + a'c' + b'c' + abc



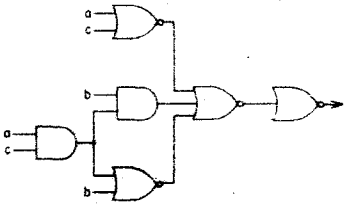
NO		FUNCTION
100	206	a'b'c + a'bc' + abc



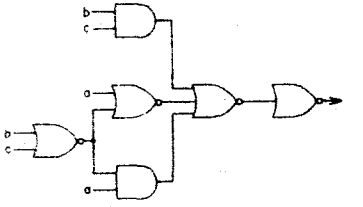
NO		FUNCTION
101	151	a'bc + ab'c + abc' + a'b'c'



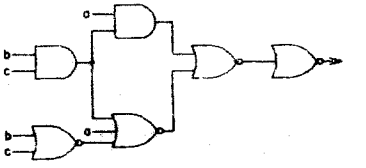
NO		FUNCTION
102	227	ab' + a'c' + b'c' + abc



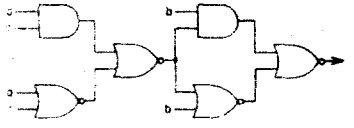
NO		FUNCTION
103	236	a'c + bc + a'b + ab'c'



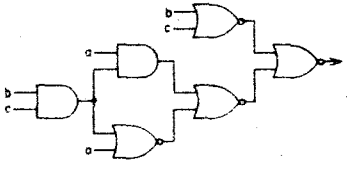
NO		FUNCTION
104	206	ab'c + ab'c' + abc



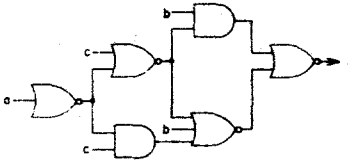
NO		FUNCTION
105	226	abc + ab'c' + a'b'c + a'b'c



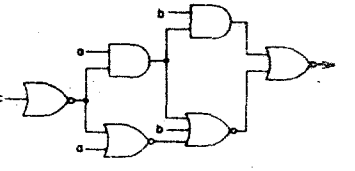
NO		FUNCTION
106	206	a'b'c + a'bc' + abc



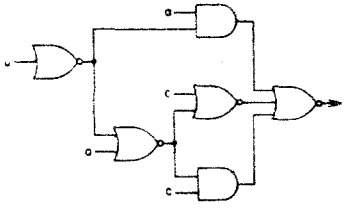
NO		FUNCTION
107	236	a'c + bc + a'b + ab'c'



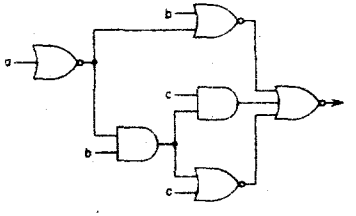
NO		FUNCTION
108	236	a'c + bc + a'b + ab'c'



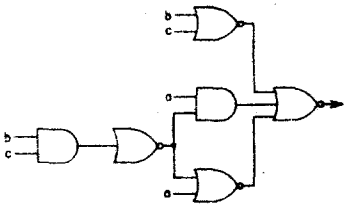
NO		FUNCTION
109	206	a'b'c + a'bc' + a b c



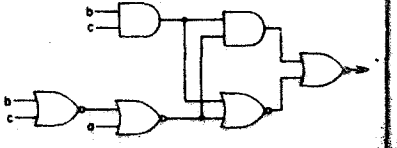
NO		FUNCTION
110	206	a'b'c + a'bc' + abc

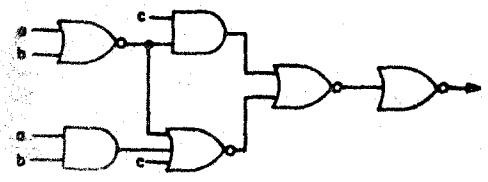


NO		FUNCTION
111	206	a'b'c + a'bc' + abc

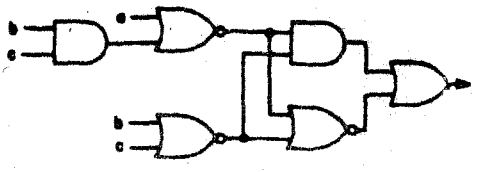


NO		FUNCTION
112	206	a'b'c + a'bc' + abc

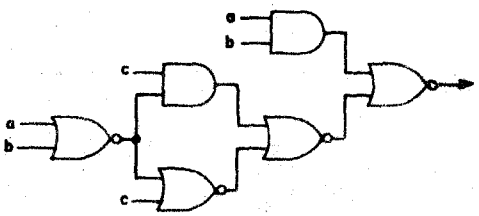




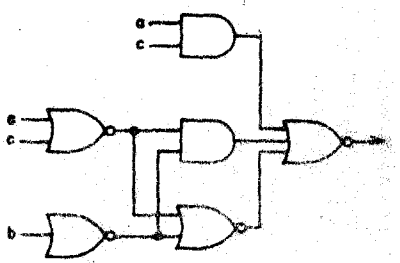
(1)



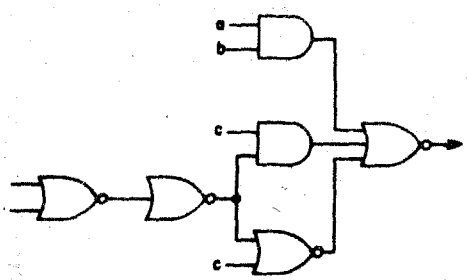
(2)



(3)



(4)



(5)

Fig.A1. Optimal networks of function 026 when the maximum fan-ins is limited to three.